A Developmental System For Organic Form Synthesis

by

Benjamin Porter, BCompSc (Hons)

Thesis

Submitted by Benjamin Porter

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Supervisor: Assoc. Prof. Jon McCormack

Associate Supervisor: Dr. Alan Dorin

Clayton School of Information Technology
Monash University

June, 2011

Contents

List of Figures

List of Tables

Supplementary DVD

The DVD attached to the back cover contains animations produced with the system described in this thesis. The system produces continuous sequences of growing forms, and as such, it is best appreciated and understood by watching these animations. The text will reference specific chapters of the DVD when the animations complement the discussion. The references to this material are of the form "see animation SDS2/3", which refers to chapter 3 in sequence SDS2 on the DVD. The DVD contains two sequences: the 2D animations (SDS2) and the 3D animations (SDS3). The chapters of each sequence are listed below.

SDS2 Animations

1. Introduction
2. Interaction with an s-morph
3. Limb growth
4. Worm
5. Starfish
6. "It Looks Like An Echinoderm"
7. Another limbed form
8. Segmentation
9. Luminous urchin

SDS3 Animations

1. Introduction
2. Physics tests
3. Failed limb experiments
4. Single limb growth
5. Six-limbed orb
6. Octopus
7. Many limbs
8. Attraction
9. Not quite a starfish
10. Starfish
11. Curling limbs

A Developmental System For Organic Form Synthesis

Benjamin Porter, BCompSc (Hons)
benjamin.porter@monash.edu
Monash University, 2011

Supervisor: Assoc. Prof. Jon McCormack
Associate Supervisor: Dr. Alan Dorin

Abstract

In many applications of computer graphics it is necessary to model natural forms. Worlds populated with both familiar and exotic flora and fauna, for example, are commonplace in animated films and computer games. Modelling organic forms using traditional computer-aided design or animation tools, however, is often a tedious and time-consuming process, particularly when constructing forms that grow and develop large amounts of complexity from simple beginnings.

Procedural modelling, an alternative to the traditional, or "manual", approach to geometric modelling, seeks to address the problem of model complexity. Under this approach, a user specifies a procedure — often with associated parameters and initial conditions — which the computer then executes in order to construct a model. Procedural modelling systems are capable of generating extremely complex structures, such as sprawling landscapes, cityscapes, and forests. If biological or organic shapes are to be modelled, it is natural to consider the processes that occur within biological development. Developmental systems, based on aspects of biological development, have emerged as a powerful technique for generating a rich variety of organic forms; however, there remains a number of forms that existing systems are unable to generate effectively.

This thesis introduces the *Simplicial Developmental System* (SDS), a system capable of automatically generating developing organic forms that are difficult, or impossible, to create using existing methods. These forms can be characterised in natural language as "organic, smooth, soft, squishy, and modular". SDS integrates a number of elements, such as cellular behaviour, soft-body mechanics and morphogen diffusion. These elements operate on a unified, adaptive, geometric representation of volumetric form – the simplicial complex – that is embedded within a physically simulated environment. The thesis presents two instances of SDS: a system for generating 2D forms with a triangular mesh representation, and a system for generating 3D forms with a tetrahedral mesh representation. Modelling biological development on these representations raises a number of questions; for instance, how does the division of a cell modify the tetrahedral mesh? This research offers a solution to this and other technical challenges.

The capabilities of SDS are exhibited in a number of experimental results presented in the thesis. These experiments demonstrate how SDS can be used to generate continuous sequences of developing organic forms. Two biological models of growth were successfully implemented in the 2D system: limb bud development in chicken embryogenesis and antero-posterior segmentation in *Drosophila melanogaster*. Experiments with these models demonstrate how cells in SDS can coordinate their behaviour in order to generate higher-order structures and patterns. This thesis also presents a number of experiments performed with the 3D system, demonstrating the effectiveness of SDS in automating the generation of complex 3D geometric models with organic features such as smooth surfaces, modularity, environment-sensitivity, similarity with variation, and elastic deformation.

A Developmental System For Organic Form Synthesis

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Benjamin Porter
June 30, 2011

Acknowledgments

It is my pleasure to thank those who made this thesis possible.

I wish to thank my supervisors, Jon McCormack and Alan Dorin, who initiated my PhD project, and whose constant enthusiasm and encouragement have supported me over the years. I am indebted to Jon, who always helped me to see the big picture, and who inspired and motivated me to persist, especially throughout the thesis-writing period. I am also thankful to Alan, whose attention to detail and interesting ideas have been essential to the development of this research. I have been fortunate to have great supervisors, who are now great friends.

I am grateful to all my colleagues at the Centre for Electronic Media Art: Jon McCormack, Alan Dorin, Alice Eldridge, Aidan Lane, Ollie Bown, Taras Kowaliw, Palle Dahlstedt, Mitchell Whitelaw, Gordon Monro, Troy Innocent, Mark Guglielmetti, Peter Mcilwain, Nick Meinhold, and Bart Veldstra, for providing years of interesting discussion and collaboration. The CEMA "make-it days" will be a cherished memory. These informal events saw everyone in a coding and lamington-eating frenzy, racing to build an interesting system before the day was out. My other fond memories of CEMA are hiking through Tasmania and travelling to Dagstuhl, Germany, for a week-long seminar on creative computing where I got to meet pioneers of the field. I would also like to convey my thanks to Bart Veldstra. During his stay at CEMA he developed a program that allowed my software to communicate with Blender, a 3D modelling program. This greatly simplified the process of setting up experiments and rendering results, thus speeding up the rate at which I could conduct my research. Without his help I would not have made so much progress.

I would also like to thank my fellow postgraduate students, for making Monash a fun and stimulating environment to be in: Fabian Bohnert, Greg Paperin, Arun Mani, Marc Cheong, Julie Bernal, Minh Duc Cao, Yee Ling Boo, Cagatay Goncu, Dhananjay Thiruvady, Kerri Morgan, Rebecca Robinson, Chris Mears, Mauro Bampo, Marsha Minchenko, Cameron McCormack, Mija Amir, and Yanir Seroussi. Many of these people have become my good friends, and I will treasure the memories of chatting in the coffee room, having beers on the balcony, playing badminton, talking about matroids and memes, and jogging around campus. I would also like to thank Graham Farr and the combinatorics research group for inviting me to take part in their regular research meetings. I have always had an interest in pure mathematics and these meetings were a lot of fun for me.

I am grateful to the administrative staff in the Clayton School of Information Technology. My time at Monash, whether studying, tutoring or lecturing, ran very smoothly due to the efforts of all these people.

I wish to thank my entire extended family for providing a loving and fun environment for me to grow up in, and for supporting my many years as a student. I want to thank my father, who introduced me to the wonders of computer programming when I was six years old. And I want to thank my mother, who has nurtured my interest in learning, and lovingly supported the decisions I've made in life.

Finally, I'd like to express my deepest gratitude to my wife, Steph, for loving and supporting me these past years. Without her support I would not have completed this thesis, and, with this research over, I now hope that we can spend more time together.

Benjamin Porter

Monash University
June 2011

Chapter 1

Introduction

The field of computer graphics plays a vital role in many facets of contemporary culture and industry, including film, computer games, simulation, and visualisation. The desire for more complex, realistic, and detailed models has driven research into texture synthesis, rendering techniques, and geometric modelling. Modern games and cinematic effects, for example, often require complex and detailed natural forms (including flora and fauna) as part of the simulated environment they create for audiences.

Generating these forms presents an enormous challenge for the geometric surface modelling techniques found in the current generation of graphics and animation software. The structures of most natural forms are not readily modelled using regular geometric primitives (such as cubes, cylinders, cones, etc.). The intricate and complex details found in organic shapes are difficult to reproduce by manual editing of primitives, even when using more complex representations such as subdivision surfaces (Catmull and Clark, 1978) or interpolative methods, such as NURBS (Forrest, 1980). Moreover, natural forms are not static, undergoing geometric and topological change as development occurs. Automated techniques are thus becoming a necessary component of the modelling process, if we are to achieve the developmental detail, geometric complexity and topological changes exhibited in nature.

For some time, a promising approach in computer graphics has been to "offload" the task of generating this detail and complexity to the computer. This is often known as *procedural modelling*, because the user specifies some procedure — along with associated parameters and initial conditions — which the computer then executes in order to build the model (see e.g., Ebert et al., 2003). Typically, the user specifies the form at a more simple, abstract level and the computer "fills in the details" in order to generate the variation and complexity required.

Organic forms seen in nature are created through a complicated process of biological development, many details of which are still not completely understood. However, enough is known about development at the cellular level to make realistic modelling a possibility. It makes sense then, that by studying and simulating aspects of biological development, we should be able to generate organic structures, and indeed this has been demonstrated for a wide range of shapes and forms (Prusinkiewicz and Lindenmayer, 1996; McCormack, 2005; Greene, 1989; Combaz and Neyret, 2006).

This thesis presents a new developmental system, the *Simplicial Developmental System* (SDS), designed specifically to generate a variety of organic forms which are difficult, or impossible, to generate with existing systems. Through simulation of developmental processes at an abstracted cellular level, and in a physically embedded environment, SDS allows the user to successfully model the growth and development of organic form in both two and three-dimensional space.

1.1 Developmental Modelling

There are numerous software packages available for 3D modelling, ranging from sophisticated modelling and animation packages (e.g., Blender[1]) to low-polygon modelling programs for use in computer games (e.g., MilkShape[2]). These packages all offer an approach to modelling in which a user, or *modeller*, manually manipulates geometric primitives, or applies a series of specialised operations, to construct a geometric model. The most common approach, polygonal surface modelling, involves direct manipulation of the vertices, edges, and faces of a polygonal mesh, as well as the application of geometric operations such as subdivision, face extrusion, and vertex merging (Foley et al., 1990, §11). This approach to modelling is extremely flexible, however the creation of complex geometry is typically tedious and requires a talented and highly specialised modeller, often with many years of experience in modelling with such systems.

Generative design is a procedural technique that offers an alternative paradigm to manual modelling, in which a geometric model is constructed not by a user, but by an algorithm (McCormack et al., 2004). Generative 3D modelling systems can create very complex, often environmentally-sensitive, structures from proportionally simpler representations. Moreover, by altering the input to the system, variations on a theme can often be generated. Generative systems offer a unique approach to design, requiring the user to consider the process of form generation, rather

[1]`http://www.blender.org/`
[2]`http://chumbalum.swissquake.ch/ms3d`

than the instantiated form itself. Generative modelling systems have been used in architectural design, digital art, and within film and computer game design. Commercial generative design software is available for generating plants, trees and forests (e.g., Xfrog[3]), entire cities (e.g., CityEngine[4]), and sprawling landscapes (e.g., Terragen[5]).

When modelling biological or organic shapes, it is natural to consider the processes that occur in real biological development. The construction of a complex, multi-cellular organism from a single fertilised cell is a powerful phenomenon. The study of the initial stages of development, *embryology*, can be traced back to Aristotle in the fourth century B.C. (Gilbert, 2006, p5). Modern fields far removed from biology are now beginning to appreciate the power of developmental processes and are applying them to design problems beyond biological simulation. Interest from theoretical (Lindenmayer, 1967), applied (Stanley and Miikkulainen, 2003) and creative (Prusinkiewicz and Lindenmayer, 1996) domains have led to the study of *developmental systems*: abstract systems that attempt to achieve the same representational efficiency, generative capability and robustness of organism development.

Developmental systems have also been studied in computer graphics in order to autonomously create complex forms and textures. There are many established systems in computer graphics that incorporate aspects of biological development for modelling organic shape, ranging from abstract symbol manipulation systems to full physical and biological simulators (Prusinkiewicz and Lindenmayer, 1996; McCormack, 2005; Greene, 1989; Combaz and Neyret, 2006).

Perhaps the simplest abstraction of development is to discard the concepts of physics, geometry, environment, and space, by modelling development purely at a symbolic level. L-systems (discussed in §2.1.1) take this approach, in which development is modelled as a discrete sequence of events acting on a string of symbols. L-systems are conceptually elegant and fast, and have been used extensively to model the growth of plants, trees, flowers, and other biological structures (Prusinkiewicz and Lindenmayer, 1996). Under the L-system methodology, a 3D biological structure is not directly modelled, but rather it is constructed from the symbol string. The symbolic nature of the system allows for efficient developmental simulation, however, due to the one-way geometric construction process, spatial and other interactions of the geometry and its environment cannot be fed back into the developmental system.

[3]`http://www.xfrog.com/`
[4]`http://www.procedural.com/`
[5]`http://www.planetside.co.uk/`

Some developmental systems model space, and allow spatial interactions to occur between components. By allowing spatial interactions and environmental conditions to feed back into the developmental rules, Voxel Automata (described in §2.1.3) generate amazingly complex structures (such as creeping vines) that grow around each other, spread and grow over walls and buildings within the environment, and seek out sunlit regions. This system demonstrates the behavioural property of many generative systems: the *emergence* of complexity from simple growth rules combined with feedback from the environment into the developing system. A limitation of this system is that, due to the static nature of the representation (vine segments cannot move after being placed), some physical phenomena cannot be modelled; for example, a branch cannot push another branch out of the way. The biological structures in VA are represented as growing lines, snaking and branching through space. This representation is efficient and suited to modelling vine growth, however it means that VA is incapable of modelling more topologically expressive structures, such as 3D surface and volumes.

Modelling development at the level of surface geometry allows a wide range of organic and smooth 3D forms to be autonomously generated (Kaandorp and Kübler, 2001; Smith, 2006; Combaz and Neyret, 2006). Modelling biological structures with a mesh is, however, considerably more difficult than other approaches, such as L-systems. One problem is that modelling cell division requires an algorithm to modify a mesh in an appropriate manner. Vertex-Vertex systems (VV) address this problem by providing a language for manipulating a mesh from a local perspective, ideal for modelling developmental events such as cell division (VV systems are discussed in §2.2.3). VV systems support a range of surface-based developmental models, demonstrated, for example, by Smith's model of the growth of a branching plant (Smith, 2006, Chapter 7).

The organic forms we see in the natural world arise not only through biological development, but are also shaped by physical forces. It has been demonstrated that modelling the physics of matter within a developmental system can greatly increase the realism of the generated forms, perhaps most elegantly illustrated by the complex folding forms generated by Combaz and Neyret's system (discussed in §2.2.4). In their system, a geometric surface, modelled as an elastic "shell", grows in response to hot-spots painted on the structure. The growth of the structure against the constraining physical forces results in the emergence of beautiful organic structures with complex geometric texture.

There are a number of ways biological development can be modelled for application in computer graphics. Depending on the "kind" of form a user is trying to generate, some aspects of development may be more or less important. The next section discusses the kind of form this thesis is concerned with.

1.2 Automated Generation of Organic Forms

While a wide range of natural forms can be generated with existing techniques, there are still many kinds of natural forms these methods cannot model effectively. The aim of the research presented in this thesis is to design a system for the automated generation of a class of form that can be described in natural language as "organic, smooth, soft, squishy, and modular" (such as the *Siphonophorae* illustrated in Figure 1.1). In order to efficiently model the development of such forms, a number of problems need to be addressed.

First is the problem of *spatial interaction.* A primary objective of this research is to be able to model the physical deformations that occur when parts of a form are constrained for space, either because there are a large number of them in a small space (such as the tubular appendages of the forms shown in Figure 1.1), or because there are environmental obstacles. Therefore spatial interactions between elements of the form need to be detected and fed back into the developmental system.

The second problem to be addressed is that of *physicality*, wherein form is shaped by deformations due to spatial interactions and other forces. Consider, for example, the tubular appendages of the forms in Figure 1.1. It is obvious that physics is playing a role in their exact shape, as they are attached to a small surface area, pushing against each other, and contorting themselves into different shapes. The intention of this research was to model deformations, such as this, that make the form appear as if it is composed of soft matter. Physical modelling in developmental systems has been investigated with appealing results (Jirasek et al., 2000; Combaz and Neyret, 2006), indicating that a model of physicality would greatly assist in the generation of the forms desired in this thesis.

Perhaps the most important consideration is: how to represent a developmental system in a way that achieves the intended modelling goals, while remaining efficient and useable. How should biological cells be modelled? Is the cell a fundamental structural unit, as in L-systems, or is a structure modelled as a continuum of cells, as in Combaz and Neyret's system? How do developmental and physical events affect the representation? Spatial interactions are required, hence there must be a spatial component to the representation. Moreover, the representation has to be capable of expressing complex surface structures, either directly as in Vertex-Vertex systems, or indirectly, as in L-systems.

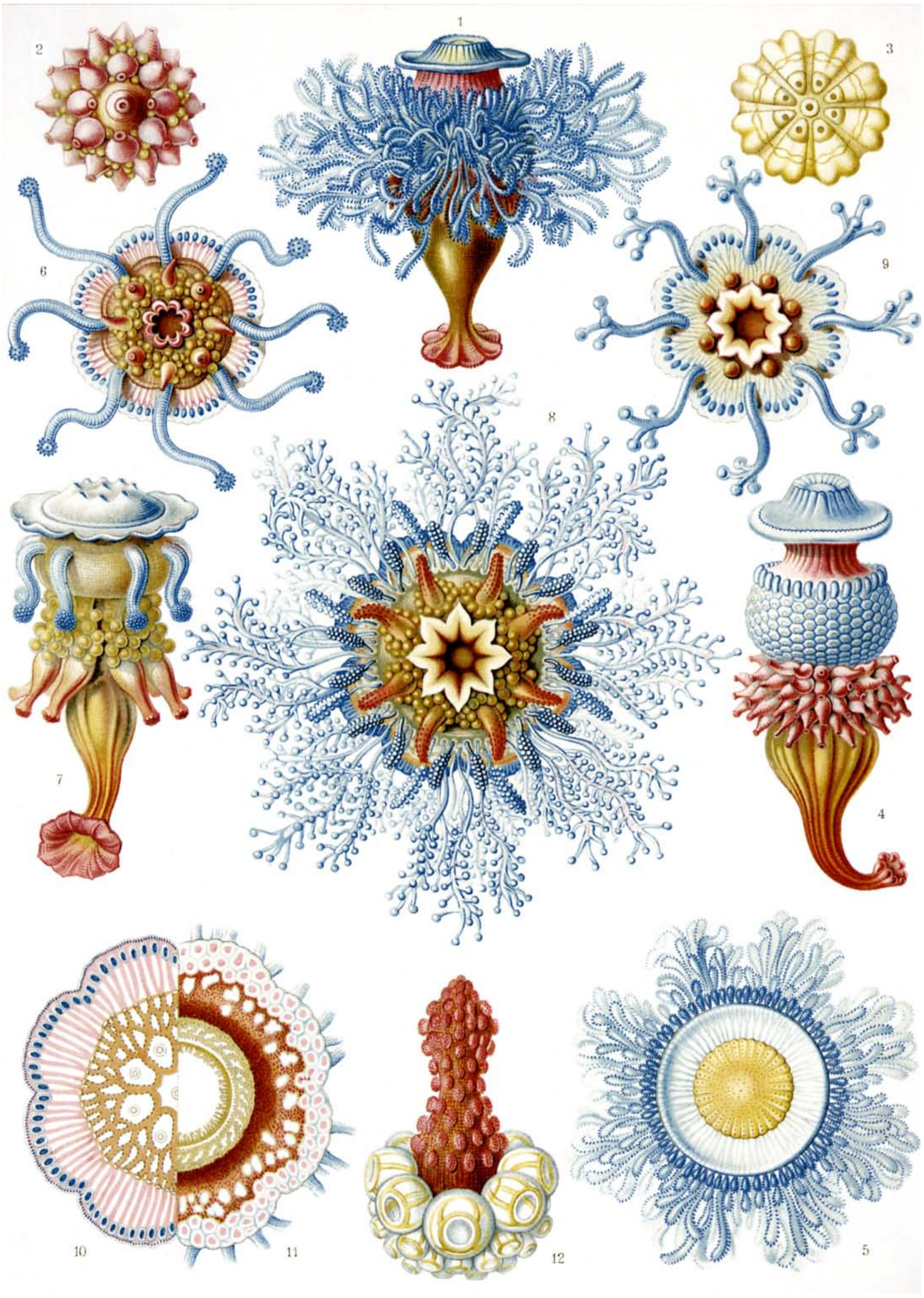


Figure 1.1: Characteristics of the "organic, smooth, soft, squishy, and modular" forms pursued in this research are nicely highlighted by Ernst Haeckel in his illustration of *Siphonophorae* (Haeckel, 1904, Plate 17).

The final problem, common to all cell-based developmental systems, is that of *emergence*. Developmental systems typically use processes that act locally, a basic example being cell division, which splits a cell into two adjacent daughter cells. A complex structure emerges through the cumulative effect of these local actions. A major benefit of this is that the amount of information required to generate a structure is considerably less than the amount required to describe the entire structure — a property known as *database amplification* (Smith, 1984). This reduces the effort on behalf of the user as it allows complexity to emerge from the repeated application of simple rules. However, producing structures from low-level processes is, in general, very difficult, due to the non-intuitive way a user must design: by specifying a process, executing the simulation, observing the results, modifying the process accordingly, and so on. If the generative system is to be useful, these considerations must be addressed.

1.3 Contributions of this Thesis

This thesis presents a new system that, by addressing the problems outlined above, is able to autonomously synthesise complex organic 2D and 3D geometry, using processes inspired by the biological development of organisms. A user of the system sets up the initial conditions of the world and specifies a model of growth; the system then simulates a developmental sequence of forms. In the 3D system, these forms can be easily converted into a polygonal surface representation common to most 3D modelling environments. Figure 1.2 presents an example of output from the system. This research has applications primarily in 3D geometric modelling, where it could be used to model creatures or organic structures for computer games, films, and other design disciplines. Another potential application of this research is in theoretical biology, as models of biological development can be simulated and tested within the system (see e.g., §5.1.2).

The key contributions of this thesis are:

- The introduction of a general simulation framework that unifies a cell-based model of development with a physical model of soft-bodied matter. The system proposes the use of an adaptive simplicial complex as a flexible spatial representation, which is embedded within an environment.

- An implementation of the framework in 2D that models a developing structure with an adaptive triangular mesh embedded in an environment. Implementation specific details of the system, such as the physical and morphogen models are presented. In addition, algorithms are presented and compared for modelling cell division and adaptation to cell movement.

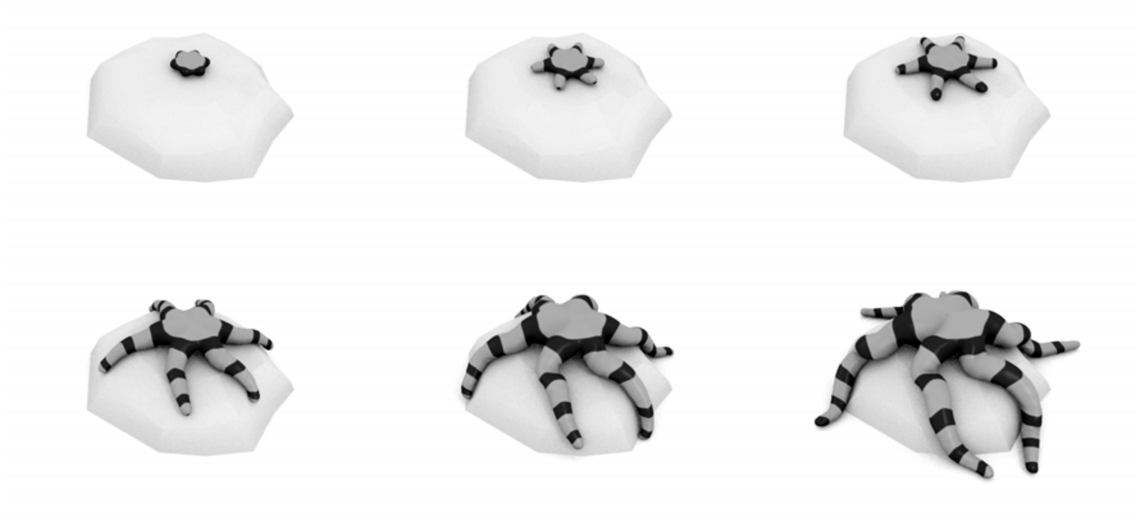


Figure 1.2: (left to right, top to bottom) A sequence of 3D geometric models of a striped starfish generated by the system introduced in this thesis. The system simulates the growth of the starfish and the generation of the stripe pattern. The system allows spatial interactions to occur between a developing structure and its environment, demonstrated here by the starfish geometry following the contour of the rock (see §7.5).

- Experiments with the 2D system that show how macro-scale structures can emerge by coordinating cell behaviour. This is demonstrated through the implementation of two biological models of development: a model of limb bud development based on early chicken embryogenesis and a model of *Drosophila* stripe formation.

- An implementation of the framework in 3D that models a developing structure with an adaptive tetrahedral mesh embedded in an environment. This system is the first developmental system to model a biological structure with an adaptive tetrahedral mesh governed by a physical model. A range of problems are addressed, including: how cell division modifies a tetrahedral mesh, how a tetrahedral mesh can adapt to the movement of cells, and how to model of soft-body physics and morphogen transport in 3D.

- Experiments with the 3D system, showing the generation of a range of complex, organic, limbed forms. Features of the system such as modularity, environmental interaction, and the use of timers to activate sequences of events are demonstrated.

- A discussion of this new approach to form generation, with numerous suggestions proposed for future work and strategies for further advancing this field.

1.4 Thesis Outline

The remainder of this thesis is structured as follows:

Chapter 2 examines the seminal work and state of the art in developmental modelling of organic structure and form. This review identifies the important features of existing systems that influenced the original research presented here.

Chapter 3 introduces the Simplicial Developmental System (SDS), a novel system for generating organic forms in two and three dimensions. This chapter introduces the concept of the *simplicial-morph*, or *s-morph* , which models a collection of cells connected together with a simplicial complex. The cells of the s-morph behave autonomously, executing cell programs in parallel, and can choose to grow or divide, thus modifying the s-morph. Cell division is the primary mechanism for adding more complexity to an s-morph. This complexity is shaped by a mechanical model of soft matter that ensures an s-morph has an organic appearance. The remainder of the thesis then covers the technical issues surrounding the implementation of SDS in two and three dimensions, and demonstrates the use of SDS to produce organic structures.

A 2D implementation of SDS is presented in Chapter 4. In 2D, an s-morph is a collection of cells in a 2D plane, connected together with a triangular mesh. Triangulated graphs have been used to model developmental systems for at least three decades (Matela and Fletterick, 1979); however, this research differs because its purpose is to explore principles of form generation that could be generalised to 3D. This chapter discusses the specifics of modelling cell division, cell movement, soft-body physics, and morphogen simulation in two dimensions. Having discussed the technical details, Chapter 5 presents the results of experiments with the 2D system. These experiments demonstrate how to program the cells of an s-morph in order to generate organic limb structures and a stripe pattern. These *growth models* are heavily based on two biological models of development: limb bud development in early chicken embryogenesis, and antero-posterior segmentation in *Drosophila melanogaster*.

Chapter 6 presents an implementation of SDS in 3D. In 3D, an s-morph is a collection of cells in a 3D environment, connected together by a tetrahedral mesh. This system is the first developmental system that operates on a spatially and structurally dynamic tetrahedral mesh. Existing developmental systems that produce organic forms in 3D typically use a surface representation (Smith, 2006; Combaz and Neyret, 2002; Kaandorp and Kübler, 2001), which fails to capture aspects of internal growth and volumetric material effects. Numerous theoretical and technical challenges were encountered during the implementation of SDS in 3D. This chapter compares some models of cell division on tetrahedral meshes, presents a model of adaptive meshing driven by cell movement, describes the incorporation of soft-body simulation into SDS, and presents other details of the implementation. Chapter 7 follows by presenting some different structures created with the 3D system. The experiments show variations of the limb bud model introduced in Chapter 5, and illustrate how SDS can successfully generate complex 3D forms with organic features such as smooth surfaces, modularity, and similarity with variation, by addressing those considerations outlined in Section 1.2.

Chapter 8 discusses a number of issues surrounding the new technique. For instance, it examines the trade-offs between using different algorithms for modelling cell division — a cell division algorithm may be efficient, but if it produces asymmetric structures around the newly generated cells, then it may be difficult to use. This chapter also discusses issues concerning the software implementation of SDS. The thesis concludes with Chapter 9 which proposes a number of further developments for SDS. The research presented here is just a first step towards full biological and physical simulation of organic form for computer graphics, addressing the issues discussed in the final chapter will bring that goal closer. Finally, a number of appendices containing supporting material are attached to the end of the thesis. Notably, Appendix A contains some high-resolution images of forms produced with SDS.

Some parts of the research presented in this thesis have also appeared in the following publications:

1. a review of the state of the art in developmental modelling for computer graphics, and an outline of the initial research goals of this project (Porter, 2009a),
2. a paper describing the 2D implementation of SDS (Porter, 2009b), and
3. a paper describing the 3D implementation of SDS (Porter and McCormack, 2010).

Before SDS is presented, it is important to first consider existing developmental systems and their features or limitations that inspired the research presented in this thesis. This material is described in the next chapter.

Chapter 2

Background

The goal of the research presented in this thesis was to design a system that can automatically generate a class of organic form that is difficult, or impossible, to generate using existing systems. The purpose of this chapter is to review the existing systems which influenced the design of SDS. These systems are able to achieve some aspects of the goal, but not all. Specific features of some developmental systems are discussed later in the thesis, when closely related to concepts used in SDS.

There are numerous systems that model aspects of development, either for form generation, or to study biological systems. These systems have been reviewed extensively in the literature. For a good overview and classification of visual models of morphogenesis, see Prusinkiewicz' comprehensive review (Prusinkiewicz, 1993). From an *Artificial Embryology* perspective[1], Stanley and Miikkulainen (2003) presented a review and a classification of several developmental models. Other publications of note include Lantin (1997), Sandberg (2006) and Giavitto et al. (2002) which provide general overviews of different approaches to modelling developmental processes in simulation. Papers reviewing systems from a theoretical and computational biology perspective cover plant modelling (Prusinkiewicz, 2004), cellular automata approaches to biological modelling (Ermentrout and Edelstein-Keshet, 1993), and computational modelling of biological systems (Brodland, 2004).

The first section of this chapter reviews the seminal research into developmental modelling, including L-systems, Cellular Automata, and a model of cell layers using triangulated graphs which heavily influenced the design of the system (§2.1). This section reviews the variety of approaches possible when modelling biological development, and provides the historical context of this thesis. The second section reviews some modern developmental systems capable of generating complex organic

[1]Artificial Embryology (AE) studies abstract systems inspired by biological development and is primarily focussed on evolving complex systems, such as neural networks and robot morphologies.

3D forms (§2.2). These systems are most closely related to SDS and are capable of generating forms which partially fulfil the goals of this research. Aspects of these systems, including the spatial embedding of a form into an environment, the use of physical models, and the geometric representations of biological structure are discussed. These systems inspired the design of SDS. The concluding section summarises the key features of these systems that were incorporated into the system presented in this thesis (§2.3).

2.1 Seminal Work

Research into developmental systems has a rich history, and it is enlightening to examine a few of the seminal models of development. This research offers different perspectives on modelling developmental processes, from Lindenmayer's symbolic L-systems, through to Fleischer's physically-based multi-mechanism developmental model. This section looks at some of the seminal work in developmental systems in order to gain some insight into the fundamental ideas in developmental modelling.

2.1.1 L-Systems

Lindenmayer-systems, or *L-systems*, were introduced by Aristid Lindenmayer (Lindenmayer, 1967) in order to describe the development of multicellular organisms. Lindenmayer proposed that these systems could integrate and express many facets of development including: gene control mechanisms, cell lineages, organising centers, polarity, and allometry (Herman and Rozenberg, 1975, Preface). L-systems model cells as symbols and development with parallel *rewriting rules*. An organism is represented as a string of symbols that develops through a process of parallel rewriting. Different symbols denote different cell types or states. Developmental processes such as division, differentiation, cell death, cell movement, and cell communication are all modelled by rewriting rules.

The simplest class of L-systems are the *deterministic*, *context-free* L-systems, referred to as D0L-systems[2]. D0L-systems consist of an alphabet of symbols, an axiomatic symbol from that alphabet, and a set of transition rules. For example, a simple L-system might use the alphabet $\{a, b\}$, the axiom a, and a single transition rule, $a \to ab$. The sequence this system generates is a, ab, abb, $abbb$, $abbbbb$, $\ldots$, and so on.

[2]In D0L-systems, the zero stands for zero-sided, or context-free. The alternatives are D1L and D2L-systems which allow one-sided and two-sided context dependant rules.

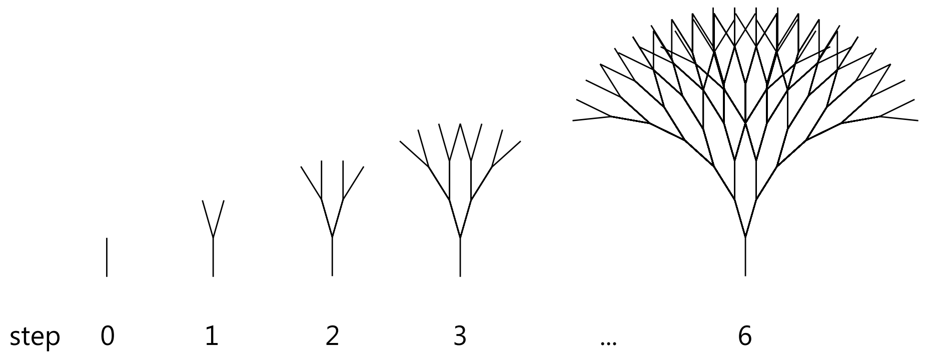


Figure 2.1: A geometric interpretation of an L-system developmental sequence. The L-system is a D0L-system with the alphabet $\{F, +, -, [,]\}$, an axiom F, and a single rule $F \rightarrow F[+F][-F]$. The sequence of strings generated by this system is F, $F[+F][-F]$, $F[+F][-F][+F[+F][-F]][-F[+F][-F]]$, and so on. These strings are interpreted by a drawing "turtle", which sits on the drawing plane and executes the following actions: 'F' move forward one unit while drawing, '$-$'/'$+$' turn left/right 16 degrees, '[' push the current position onto the stack, and ']' pop the last position off the stack and move there (without drawing).

While originally intended to model developmental biology, L-systems were also adopted in formal language theory[3] and, more relevantly, computer graphics. If we interpret the string of symbols as instructions to a geometry building machine, then a string can be mapped to a geometric structure, and a sequence of strings gives us a sequence of structures. The turtle-based machine is the most common interpretation, using the concept of the drawing *turtle*. The turtle sits in the drawing space and reads each symbol in sequence performing commands such as *draw line*, *change colour*, or *store current position*. Figure 2.1 illustrates an example of a drawing produced this way.

L-systems can also be used to generate 3D geometric structures, by associating with the turtle a position and orientation in 3D space. Parsing the derived string produced by the L-system causes the turtle to move through space, tracing out line segments, profile curves, placing geometric models, or performing other geometric actions. An example of a 3D model generated by an L-system is shown in Figure 2.2, with the corresponding L-system given in Table 2.1.

L-systems belong to the class of grammar-based approaches to procedural design, the fundament of which is the replacement rule. Other grammar-based systems include shape grammars (discussed later in §2.1.2), collage grammars (Drewes and Kreowski, 1997), boundary solid grammars (Heisserman, 1991), graph grammars

[3]By considering the set of all strings generated by a particular set of transition rules as a language we can explore the generative power of various L-systems. One particularly interesting result is that context-free Chomsky grammars form a proper subset of context-free L-systems. See (Herman and Rozenberg, 1975; Rozenberg and Salomaa, 1980, 1986) for a comprehensive survey of results in this field.

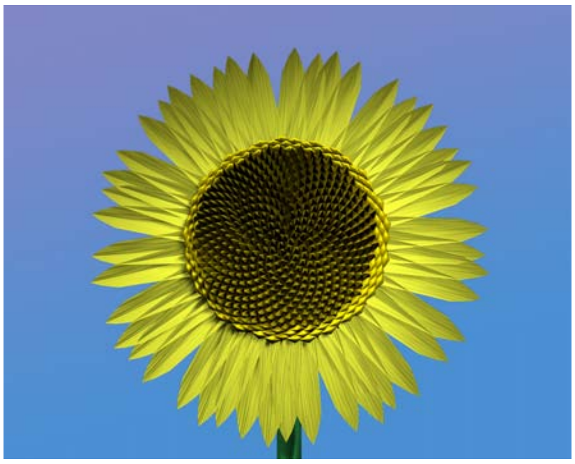


Figure 2.2: A sunflower generated by the L-system given in Table 2.1. Note how the L-system captures the symmetry of the flower elements in a succinct set of production rules. Image courtesy of Jon McCormack.

(e.g., Kniemeyer et al., 2004) and rule-based mesh growing systems (Maierhofer, 2002).

L-systems are conceptually elegant as they use a simple but expressive abstraction of development. From the perspective of computer science these systems are fast (as they operate on symbolic strings) and expressive (context-sensitive L-systems are Turing complete). As a form generating tool, L-systems are extremely powerful (Prusinkiewicz and Lindenmayer, 1996). However, to generate realistic images the original system had to be extended, for example by allowing continuous growth (Prusinkiewicz and Lindenmayer, 1990; Prusinkiewicz, 1993), physical and mechanical effects (Jirasek et al., 2000; Lam and King, 2005), explicit specification of hierarchy (Vaario, 1994; McCormack, 2005) and environmental interaction (Měch and Prusinkiewicz, 1996). These extensions are useful but depreciate the simplicity and elegance of the original formalism.

As discussed in the introduction (§1.2), physical modelling can greatly enhance the realism of models generated by developmental systems. A number of L-systems models have been proposed that integrate developmental and physical models. These range from modelling nutrient transport in a developing structure (Allen et al., 2005), to modelling the biomechanics of bending branches (Jirasek et al., 2000). The biomechanics model is particularly interesting, from a computer graphics perspective, as it produces interesting 3D plant forms with branches that bend under their own weight and are influenced by other forces such as *negative gravitropism* (the desire to grow against the pull of gravity). An interesting aspect of this system is that, by discretising the physical equations as L-system rewrite rules, the physical model can be solved using L-systems. This allows the whole system, both

Table 2.1: An L-system used to produce the 3D sunflower model shown in Figure 2.2. This *parametric* L-system associates parameters to the symbols A, C, S, f, $+$, and $\wedge$, which the L-system uses to enact parameter-dependent rules (e.g., rule 2 is executed only if the n parameter of the C symbol is less than or equal to 440). The parameters are also used by the turtle when building the geometry. The turtle interprets each symbol as specified under *symbol interpretations*.

axiom : stamen $A(0)$
production rules
$A(n) \rightarrow +(137.5)[f(n^{0.5})C(n)]A(n+1)$
$C(n) : n \leq 440 \rightarrow$ floret
$C(n) : n > 440 \ \& \ n \leq 565 \rightarrow$ floret2
$C(n) : n > 565 \ \& \ n \leq 610 \rightarrow \wedge(90)S(0.3)$leaf2
symbol interpretations
stamen, floret, floret2, leaf2: Instantiate pre-defined geometry at the current position and orientation
$f(x)$: Move forward x units
$S(x)$: Scale (uniformly) all geometry after this symbol by x units
$+(x)$: Rotate (yaw) right by x degrees
$\wedge(x)$: Rotate (pitch) up by x degrees

the developmental *and* physical aspects, to be elegantly framed within the L-system formalism.

One major feature of L-systems is the distinction between the symbolic string representing an organism and the geometry created from that string. Conceptually the developing organism exists in a one dimensional *string space*. The developmental rules operate on that string and transform it to another string. The interpretation of the string as a geometric object is a one-way procedure. This makes the developmental simulation fast but it has a major limitation. The difference between the string space and the *geometric space* raises two main issues. Firstly, the topologies of the geometry are tied to the topology of the underlying representation, for example D0L-systems are only suited to creating geometric structures with simple one dimensional topologies, like trees and other branching structures, etc. Depending on the class of object you wish to model, this may not be appropriate. This issue has been addressed by extending L-system models onto more topologically complex structures such as graphs (such as Cell systems (§4.1.1)) and polygonal surfaces (such as Vertex-Vertex systems (§2.2.3)). A second limitation is the lack of communication from the geometric space back into string space. So, for example, a model of tree development in which branches compete for space cannot be expressed as an L-system. Open L-systems seek to address this issue by adding an explicit coupling mechanism between the environment and L-system (Měch and Prusinkiewicz, 1996). This approach is effective at modelling some scenarios, however, for more complex interactions the elegance of the system breaks down.

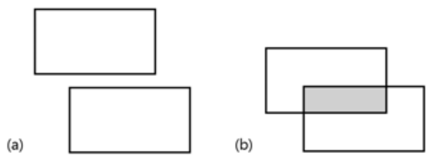


Figure 2.3: Emergent shapes in shape grammars. Translating the upper rectangle in (a) down by some distance creates in (b) an emergent rectangle (highlighted in grey). This rectangle could then be transformed by a rewrite rule.

A key goal of the research in this thesis was to reduce the gap between the internal representation of a developing structure and its visual representation (i.e., its interpretation). This gap is referred to as *embeddedness* in this thesis — a weakly embedded system is one in which this gap is large (e.g., L-systems), and a strongly embedded system is one in which this gap is small (e.g., Cellular Automata). Embeddedness is discussed in more detail later (§2.3).

2.1.2 Shape Grammars

Shape grammars (Stiny and Gips, 1972) are a formalism for describing transformations on shapes[4]. A shape grammar consists of an initial shape and a set of shape transformations. A simple example design in a shape grammar is shown in Figure 2.3(a). Shape grammar rules may move shapes, rotate shapes, add or remove shapes, or transform parts of shapes. Applying a translation rule to a rectangle in Figure 2.3(a) may result in the transformed design shown in Figure 2.3(b) (in which the top rectangle has been shifted down). As in L-systems, the complexity of a design in a shape grammar emerges through the repeated application of locally applied rules.

Shape grammars differ fundamentally from other grammar-based systems as they work directly on shapes, rather than symbolic representations of shapes. This makes shape grammar systems difficult to implement because they require shape recognition and an extremely flexible internal representation. Shape grammars are inherently more powerful than symbolic systems due to ambiguity within shapes and the emergence of new shapes (Figure 2.3). This feature makes shape grammars ideal for art, architecture and product design where emergent shapes contribute greatly to the aesthetic. Shape grammars have been applied to many domains, including Palladian houses (see Figure 2.4), Chinese Lattices (Stiny, 1977), Coffee Makers (Agarwal and Cagan, 1998), and Coke Bottles (Chau et al., 2004).

[4]A *shape* in a shape grammar is typically an object constructed from primitives such as points, lines, polygons, and curves.

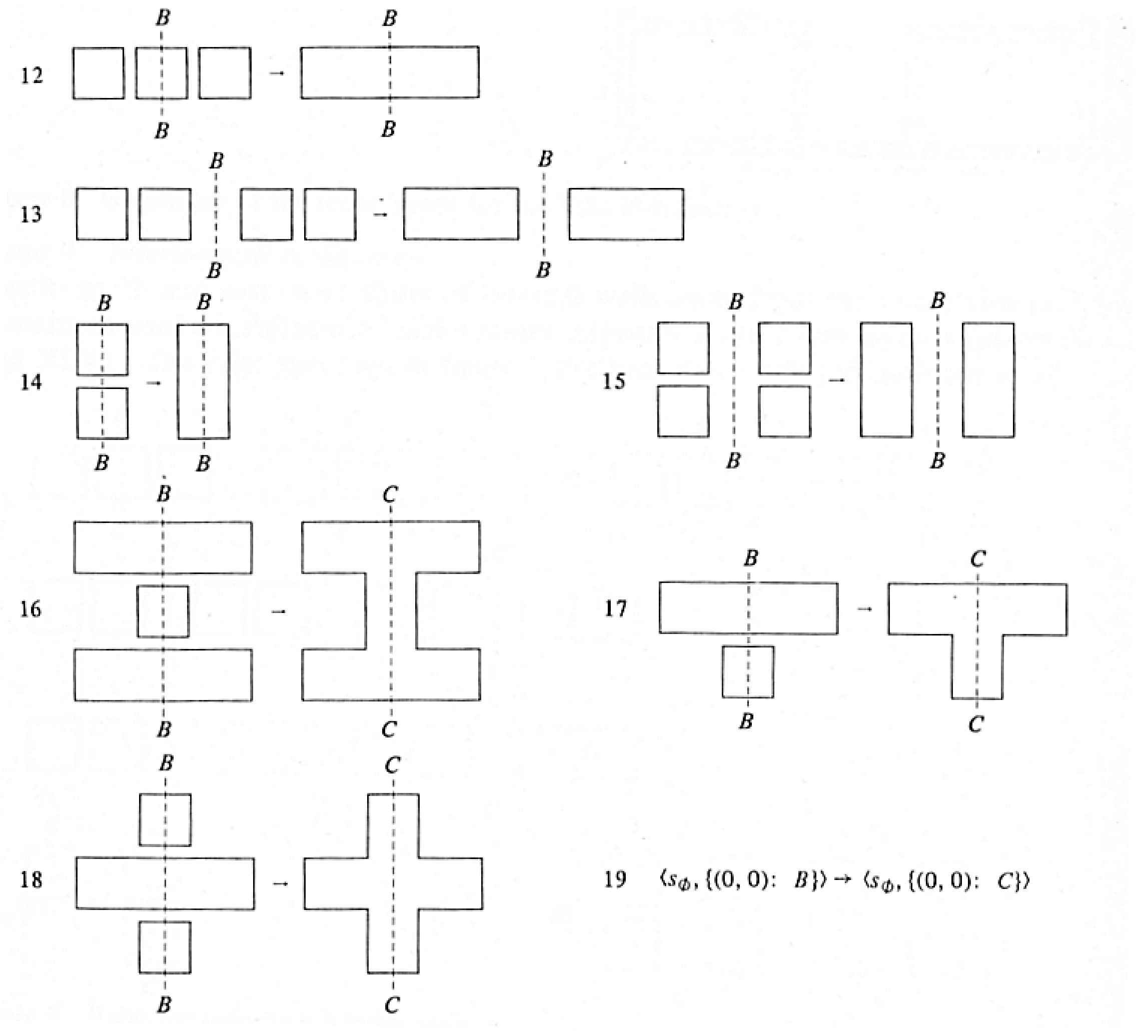


Figure 2.4: The rules for laying out rooms in an architectural shape grammar inspired by Palladio (Stiny and Gips, 1978, Figure 7). The entire model consists of 72 individual rules. Image courtesy of George Stiny.

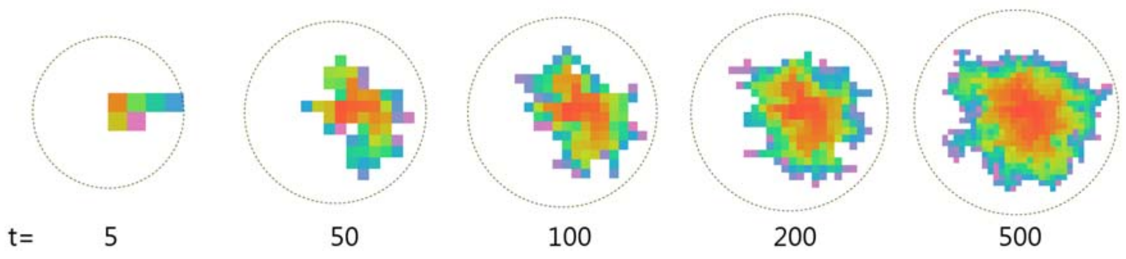


Figure 2.5: (left to right) A simulation of the Eden process over multiple time steps.

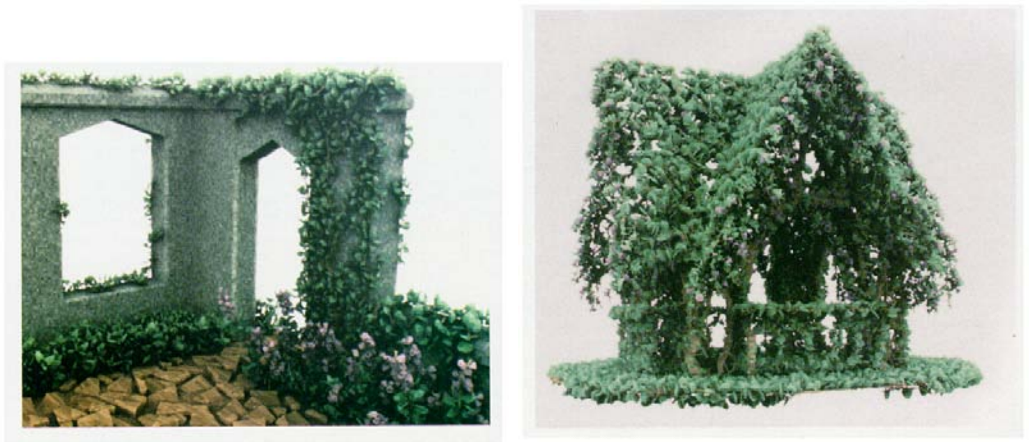


Figure 2.6: Examples of environmentally sensitive geometry generated with Voxel Space Automata (Greene, 1989, Figures 5 and 9).

2.1.3 Cellular Automata

Cellular Automata were introduced by Von Neumann (1966) and Ulam (1962) in order to study self-replicating machines and the growth of crystals respectively. CAs are a general modelling technique in which space is typically discretised into square elements. In biological modelling, three main types of CA have been identified: deterministic, lattice gas models, and solidification models (Ermentrout and Edelstein-Keshet, 1993).

Deterministic CAs (von Neumann and Ulam's original model) represent a system as a regularly connected set of finite state automata, each of which update depending on the state of neighbouring automata. In this system, a biological cell is represented as a single automaton, and there is a direct correspondence between cell and space. One of the first and simplest models of cellular growth via accretion is the Eden model (Eden, 1961). The process begins with a single cell on a regular lattice and iteratively grows the *colony* by adding a new cell adjacent to the colony at a stochastically determined site. The colony grows dense clusters with fractal surfaces (e.g., Figure 2.5).

Branching growth in CAs was originally studied by Ulam (1962). These ideas have been applied to computer graphics to synthesise three dimensional form (Greene, 1989; Kawaguchi, 1996). Greene's *Voxel Space automata* models the growth of a structure in a discretised three dimensional space. At each time step, the growing structure attempts to add a new *part* specified by a set of rules. The rules that are executed are determined by a number of conditions, including available light, proximity to some object, or the result of an intersection test. Greene observed that some complex structures arise through the interplay of probabilistic growth, environmental effects, and feedback between the growth rules and space (see Figure 2.6).

Lattice gas and solidification models discretise space but allow cells and molecules to move between sites. These models have also been used extensively to model various biological and physical phenomena. Diffusion-limited aggregation (DLA), for example, models the formation of complex aggregated structures, or "puff balls", formed in nature from the aggregation of certain kinds of solid particles (Witten and Sander, 1981). In the DLA model, a static seed particle is first placed into an environment. Particles are then introduced one at a time at the edge of the environment and randomly walk around the grid. When a walking particle comes into contact with a static particle it becomes static. The DLA model is simple and succinctly explained and yet produces complex tendrilled structures. The sorting of cell by differential adhesion is a commonly studied biological phenomenon, in which cells arrange themselves into clusters purely by choosing to adhere to certain kinds of cells. This phenomenon has been replicated in a CA model, using a two-layer CA in which cells are modelled as collections of sub-cells which move around the lattice according to physical rules (Hogeweg, 2000, 2003).

In CA-based developmental systems, the rules describing the growth of an organism are typically simple, and yet complex patterns and forms emerge. This complexity emerges because the developing structure is embedded within a spatial environment, allowing interactions to occur between initially unrelated parts of a structure. CAs are an excellent example of the power of spatial embeddedness, a feature that has guided the development of the research described in this thesis. Developmental models in CAs are typically limited to accretive (boundary) growth because of the lack of room to place new cells on the "inside" of a form. Physically-based models can help with this by allowing cells to move around based on an energy minimising term (Hogeweg, 2003). Another disadvantage of using a regular spatial discretisation is that it is tedious to represent detail over many scales. Representing two cells of different sizes, for example, necessitates splitting both cells into sub-cells. This makes this method computationally inefficient when modelling detail over multiple-scales, though this is becoming less of a factor with the recent emergence of very fast parallel computing architectures (e.g., Nvidia's CUDA[5]).

2.1.4 Matela's Triangulated Graph Model

There are two major issues with deterministic Cellular Automata. Firstly, the developing structures are bound to the topology of the underlying space. This can be alleviated somewhat by using different spatial topologies (for example a hexagonal grid). Secondly, it is difficult to model internal structural events, such as cell division. If a dividing cell is completely surrounded by other cells there is simply no room to place the two new daughter cells. These disadvantages led researchers to consider other, more flexible, representations of cellular structures, including planar graphs (Matela and Fletterick, 1979), Voronoi regions (Honda, 1978), and polygonal maps (Weliky and Oster, 1990).

Particularly important amongst these developments was the model of Matela and Fletterick, in which planar graphs were proposed to represent cellular layers, allowing a greater range of topological configurations than CAs and much more spatial freedom in which to divide and migrate (Matela and Fletterick, 1979). Matela and Fletterick's general model (Matela and Fletterick, 1979) uses a planar map to model sheets of cells: regions represent cells and edges represent cell boundaries. The resulting complex then approximates cells of polygonal shape that are densely packed (see Figure 2.7.) Allowing cells to have an arbitrary polygonal shape is the primary rationale behind the development of this model, as it provides greater flexibility over previous (hexagonal or rectangular grid-based) models (such as cellular automata) in topology, cell size and cell shape (Matela and Fletterick, 1979). Their model provided the inspiration for the representation and operations proposed in this thesis, and is considered in substantial detail later (§4.1.3).

2.1.5 Fleischer's Multi-Mechanism Model

One system that takes a more literal interpretation of biological cells is Fleischer's multi-mechanism model (Fleischer, 1995). Initially developed to study the evolution of artificial neural networks, it became a general framework for studying structure and pattern formation in collections of autonomous free-floating cells. In his thesis, Fleischer modelled many phenomena including the development of neural networks through cell targeting, the emergent formation of chains, compartments and

[5]`http://www.nvidia.com/object/cuda_home_new.html`

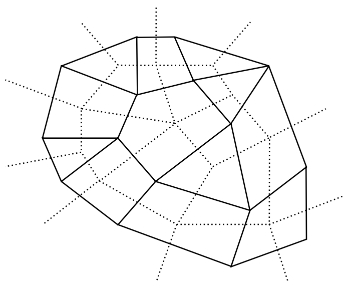


Figure 2.7: In Matela and Fletterick's developmental model cells are modelled as polygons. The developmental operations act on the cell layer topology (the dotted graph).

hierarchical structures, the emergence of structures through differential adhesion, reaction-diffusion directed growth, and segmented structures. Fleischer's model integrates mechanical interaction (collision and adhesion) and intracellular processes (transcription, regulation, kinetics, transport, metabolism, a cell surface protein model, cell lineage, cleavage plane control, neurite growth, and electrical activity) (Fleischer et al., 1995). In Fleischer's 2D model, cells are modelled as circles freely moving in a bounded plane. This model is examined in more detail later (§4.1.2).

One question that arises from applying cell-based systems to 3D modelling is: how do you generate geometry from a collection of cells? While Fleischer's primary contribution was his 2D system, he also used his system in a novel way to generate complex 3D forms. By simulating 2D cells floating on the surface of a geometric model, he then used the cell's positions and properties to generate geometric detail, such as bumps, thorns and spikes. The system presented in this thesis is also cell-based, but the relationship between cells and geometric model is quite simple: a cell is a vertex in a 3D solid mesh.

Fleischer's research can, arguably, be considered as one of the first systems in Artificial Embryology. A recent survey of this field presents a useful categorisation (Stanley and Miikkulainen, 2003). These systems are important to consider with respect to the original research presented in this thesis, because they offer insight into different models of cell behaviour and structural representations, and are considered later on (§3.3.1).

2.2 Related Work in Developmental Systems

The last section highlighted the broad spectrum of developmental systems. This section focuses on systems which achieve some aspects of the goals of this research (§1.2) and which were the inspiration for the system presented in this thesis. These systems model processes of biological development in order to generate continuous sequences of organic 3D forms, and range from an L-system inspired continuous cell-based developmental model (§2.2.1) to a biologically motivated model of coral growth (§2.2.2). As each system is reviewed, features important for modelling 3D organic smooth surfaces and limitations of the system (with respect to my research goals) are discussed.

2.2.1 Cellular Developmental Model

The Cellular Developmental Model (or CDM) by McCormack is a developmental modelling framework for application in computer animation and music synthesis (McCormack, 2005). CDM incorporates ideas from L-systems, cellular automata. and cellular programming with useful concepts such as hierarchies, asynchronous development, and a generalised context mechanism. Figure 2.8 illustrates a complex 3D model generated using this system.

CDM models a cell as an object which has a *label* that indicates its type, internal and external state, a set of production rules, and an *interpretation*. The production rules model the cell's behaviour and take the form:

$$r_i : \{\text{context}\} : \text{predicate} : \text{state calculations/actions} \tag{2.1}$$

As with L-system models, the context represents the relation of the cell in the pool to other cells, for example, the context $A(y)meB(z)$ represents the situation where an A cell is immediately to the left of the current cell, and a B cell is immediately to the right (the variables y and z are extracted from the state of A and B). The context in CDM can be more general that this, for example Euclidean distance or von Neumann neighbourhood in a 3D CA. The predicate indicates some condition that has to be true in order for the rule to be applied. Using the context above, for example, the predicate $y > k_{min}$ means that the rule is applied only if the single state variable of A is above some threshold. Finally, the third component of the rule indicates some state calculation or action to perform. The state calculation is a mathematical expression, for example $x = 2y$. The actions available are cell death, cell division, and changing a cell's type.

The interpretation of a cell is a set of instructions for constructing a representation of it. This might be, for example, an interface to a geometric algorithm that constructs a 3D mesh based on the cells and their properties. Particularly interesting is McCormack's use of generalised cylinders to generate 3D forms with smooth

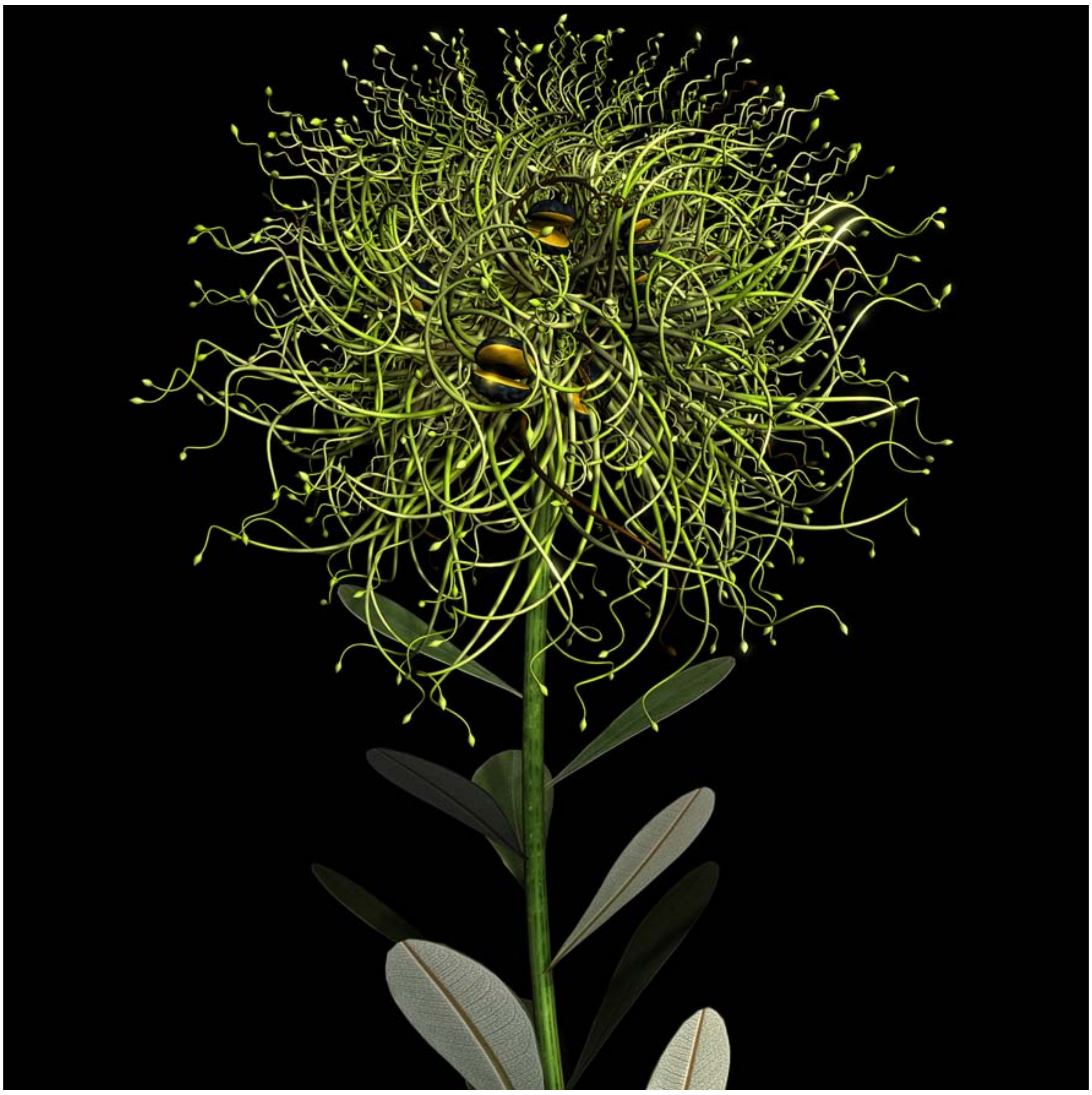


Figure 2.8: A complex structure evolved using CDM. (McCormack, J. *Morphogenesis Series #11*, Lightjet print on archival paper, 1.5m x 1.5m, 2006.) Image courtesy of Jon McCormack.

continuous surfaces. This addresses the visual anomaly of sharp creases or joins between components that occurs in geometry constructed with traditional L-systems, and offers a simple type of spatial interaction or "awareness" between modules. The generality of CDM and it's interpretation layer, however, means that is cannot model more complex spatial interactions, such as collisions between modules. Hence CDM is unsuitable for addressing the specific goals of this research, but nonetheless is a powerful method for generating complex dynamic 3D developmental forms and animations.

The CDM concepts of "cell" and "cell rules" inspired similar concepts that were incorporated into SDS. As in CDM, cells in SDS are autonomous entities which execute behaviour-defining sets of rules; however, this is where the similarity ends. A cell in CDM is an abstract module in a CDM system, and is responsible for building a geometric model via construction commands determined by the interpretation of the system — there is a clear distinction between a system of cells and the output of the system. A cell in SDS, however, is an element of a geometric structure and is tightly coupled to the geometric models produced by the system. The distinction between these systems arises primarily because SDS is designed to support a flow of information between the geometry of a form and the system that generates it, whereas CDM makes a conceptual distinction between the process of development and the modes of output produced by the system (e.g., geometric models or sound).

2.2.2 Modelling Accretive Growth

A developmental model of accretive growth that demonstrates the combination of a geometric surface-based developmental model with a physical model of nutrient acquisition and hydrodynamics was developed by Kaandorp (1994); Kaandorp and Kübler (2001). While scientific in application, their model is nonetheless useful to consider here because it simulates the growth of a surface embedded in an environment. The developmental process is initialised with a triangulated sphere. A *growth process* then repeatedly constructs a new triangulated surface around the old one. This method is particularly interesting as it does not attempt to modify the old surface but merely adds a new layer on top of it. This is conceptually appropriate for the accretive growth processes it models. One experiment performed with the system grows a surface at a rate proportional to the availability of food particles dispersed throughout the environment (Kaandorp and Kübler, 2001, §4.6.4). A fluid simulation is performed to compute the distribution of food. These experiments demonstrate that a combination of a simple growth logic, a physical model and embedding a growing structure into an environment can result in complex and organic forms. Kaandorp and Kübler's system is designed specifically for modelling accretive growth of hard structures, and it is incapable of modelling deformations due to physical collisions and the processes of development that occur inside a structures. Nonetheless, their model of growth that acts on a polygonal mesh is interesting, and is considered in more detail later when discussing the representation of 3D forms used in SDS (§6.1.2.1).

2.2.3 Vertex-Vertex Systems

Smith's *Vertex-Vertex systems* (VV) are a general purpose modelling framework that can be used to model polygonal meshes (2-manifold topologies) which are both spatially and structurally dynamic (Smith, 2006). The VV algebra and language provide a method for manipulating meshes from a local perspective, i.e., without the need for a global indexing scheme. For example, the instruction `splice` *x* `after` *a* `in` *v* indicates that a new vertex x is to be inserted into the neighbourhood of v *after* a in a clockwise order. The ability to specify transformations locally is the key to VV, and makes it an ideal system for modelling biological development.

Geometric algorithms can be specified using the VV language. These can range from adding a new vertex on an edge (Smith, 2006, Algorithm 3.1) to complex surface subdivision schemes (Smith, 2006, Chapter 4). VV can also be used to implement physically-based models of growth, as demonstrated by Smith's simulation of the growth of a branching plant that incorporates a simple mechanical model (Smith, 2006, Chapter 7). The result is an animated smooth polygonal mesh of a plant-like form with a nice stem-to-branch geometric interface (Figure 2.9). Smith models the surface of the structure as a mass-spring system with an additional *pressure* term that provides an outward pushing force. Extra rules such as a simple tropism model and reducing pressure at the growing tips of the primordia contribute towards the final result.

The flexibility of the VV approach in modelling development either from a cell level (as in Smith, 2006, §7.2) or a mesh level makes it useful in a wide range of developmental modelling scenarios, however, the restriction of VV's representation to surface meshes causes some problems. Firstly, VV is incapable of modelling growth or other developmental events that occur internally. Secondly, being surface-based also means that the physical model has to use a very rough approximation to keep the form "solid" (Smith uses a *pressure term* which applies a normal force on the vertices.)

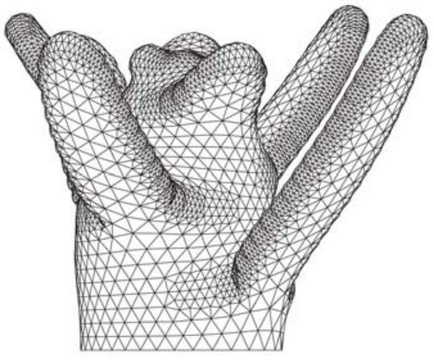


Figure 2.9: Geometry synthesized by a model of phyllotaxis implemented in Vertex-Vertex systems (Smith, 2006, Figure 9.4). Image courtesy of Colin Smith.

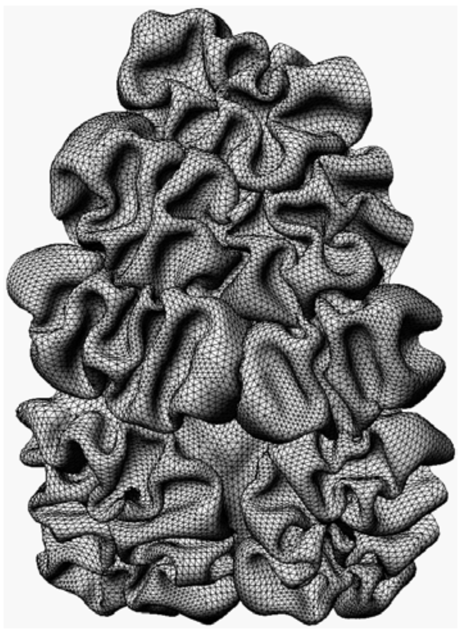


Figure 2.10: A complex folding surface created using Combaz and Neyret's physically-based growth system (Combaz and Neyret, 2002). Image courtesy of Fabrice Neyret.

If only surface-level events are to be modelled then VV systems are a sophisticated and practical 3D generative modelling tool. An early strand of the research presented in this thesis considered extending VV to operate on solid forms, thus addressing some of these issues. Unfortunately (for reasons discussed in §6.1.2.3) the topological and algebraic basis for VV cannot be generalised into 3D topologies, and so a different approach was sought. Vertex-Vertex systems are the most closely related (of existing systems) to the 3D model presented in this thesis, and are revisited later when discussing the details of the 3D system (§6.1.2.3).

2.2.4 Semi-Interactive Morphogenesis

An alternative approach to modelling individual cells within an organism is to model an organism as a *continuum* of cells and consider the aggregate effects of cell actions. One system that takes this approach is Combaz and Neyret's semi-interactive morphogenesis system (Combaz and Neyret, 2006, 2002). In their system, a user paints a texture onto the surface of a triangular mesh, stimulating local growth and change. The texture specifies the desired state of local mesh properties, such as edge length and vertex curvature. The mesh elements are elastic and a solver iteratively reduces the energy of the system. The physical model incorporates the continuum

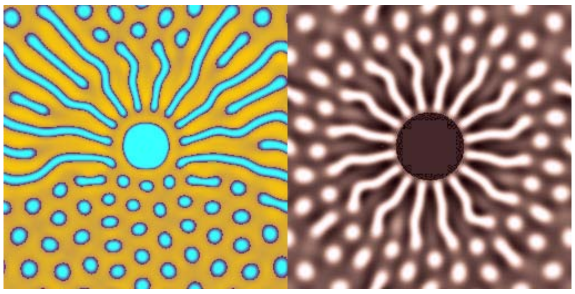


Figure 2.11: Two complex reaction-diffusion patterns (Sanderson et al., 2006, Figure 15). Image courtesy of Allen Sanderson, copyright A K Peters, originally published in Journal of Graphics Tools Vol 11, No. 3.

mechanics theory of thin shells (discussed later §6.1.2.2). The resulting forms have a smooth, soft, organic appearance that would be extremely tedious to build by hand (Figure 2.10).

Combaz and Neyret's work elegantly demonstrates that a model of elastic surface growth can produce quite complex and organic geometry. The system, however, is not entirely automated, and control is performed primarily by a user interactively painting hot-spots on the growing surface. Procedural control of these hot-spots were explored (Combaz and Neyret, 2006); however these results are preliminary and it is not clear as yet if this approach applies generally. The semi-interactive morphogenesis approach works extremely well for generating the geometric complexity of folds and geometric "texture", however as it lacks a mechanism for coordinating more complex structures, it is limited to producing local geometric features, and hence is unsuitable for generating modular forms.

2.2.5 Morphogen Coupled Growth

In 1952, Turing showed that complex patterns can arise from simple systems of interacting chemicals or *morphogens* (Turing, 1952). Turing postulated that these *reaction-diffusion* systems exist in biological systems and assist in coordinating the spatial expression of genes.[6] These systems have been investigated in depth (Meinhardt, 1982, 2003) and have also been applied to the creative synthesis of complex patterns (Figure 2.11) (Turk, 1991; Sanderson et al., 2006).

Reaction-diffusion models have been coupled to geometry to investigate morphogen-directed growth (Harrison et al., 2001; Leung and Berzins, 2003; Cummings, 2001). From a creative perspective these systems illuminate how growth can be coordinated

[6]The existence of reaction-diffusion in biological systems has been recently confirmed (Sick et al., 2006).

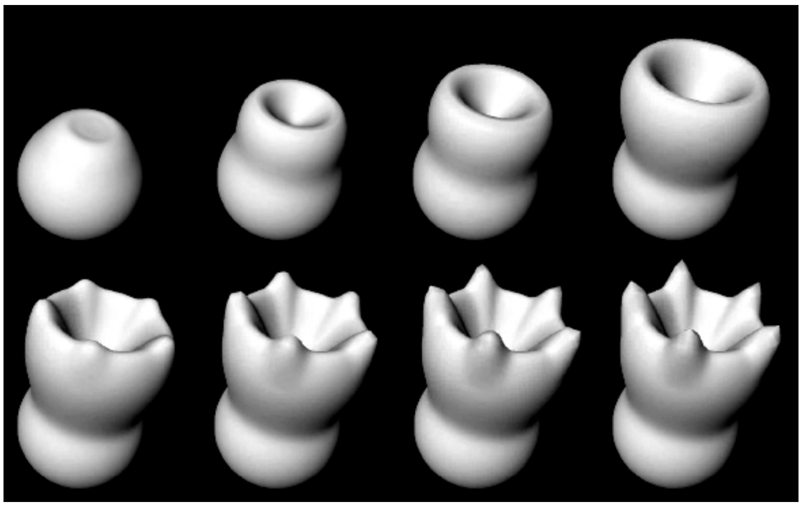


Figure 2.12: (left to right, top to bottom) A model of a developing polyp by Leung and Berzin (Leung and Berzins, 2003, Figure 9).

in a continuum model. Leung and Berzins (2003) present an interesting computational model of development. They link concepts of cell bio-chemistry and surface deformation to model simple developing shapes (Figure 2.12). Their model uses a reaction-diffusion simulation of morphogens on a surface. The surface curvature is locally altered by the presence of morphogens, thus tightly coupling the surface geometry and cell chemistry. The application of these systems to the creative modelling of organic form is hindered because of a number of reasons: the reaction-diffusion process is notoriously hard to control, the geometric representations prevent topological changes, and growth is limited to the surface. These systems successfully demonstrate that chemical patterning coupled with surface growth can produce interesting organic forms, and, like Combaz and Neyret's system (§2.2.4), show that a model of material physics offers a powerful mechanism for generating organic complexity.

2.3 Discussion

As shown in this chapter, there are many ways in which biological development can be modelled in order to generate forms and patterns. The research of this thesis is concerned with modelling the development of complex 3D organic forms. A number of systems are capable of this (§2.2), however, none of these systems can generate forms of the type outlined in the previous chapter, i.e., soft, squishy forms with smooth surface that spatially interact. Features of these systems which seem critical to achieving the goals of this research are: spatial embeddedness, physicality, and representation. These features were considered when designing SDS and are explained below.

Embeddedness The embeddedness of a system measures the difference between the internal representation of a structure and its visual representation, subject to an external viewer. A strongly embedded system is one in which the developing structure exists within the space it is being viewed in, and the more embedded a system is, the less visual *anomalies* will occur. L-systems and the Cellular Developmental Model are both weakly embedded. In the L-system shown in Figure 2.1, for example, an anomaly occurs where two branches (straight lines) can intersect in space. The line intersection condition cannot affect the development of the structure as the turtle-based mapping is a one-way process.

CAs, on the other hand, are strongly embedded, as the cells are typically represented directly within a grid, often with coloured cells distinguishing cell types. As CAs most elegantly demonstrate, embedding a developing structure within an environment adds considerable complexity and an additional facet of control (often at the expense of more direct modes of control §8.2). A spatially embedded system supports spatial interactions between parts of a structure (e.g., Ulam, 1962). Kaandorp and Kübler (§2.2.2) showed that by distributing nutrients through an environment with a model of fluid flow a surface could grow in an environmentally dependent manner. One goal of the research presented in this thesis was to model spatial interactions between the geometric elements of developing forms; and thus SDS is strongly embedded, with respect to 2D and 3D geometric models.

Physicality This chapter reviewed some systems which demonstrate how a generative model that incorporates physics can lead to natural looking forms. Combaz and Neyret's system (§2.2.4) used a physically-based elastic thin shell model coupled with a growth mechanism to achieve forms with complex smooth folds. The geometric texture is a balance between procedural specification and physical simulation, resulting in the surface adopting a generally smooth organic shape. A physical model of material acts as a constraint on a generative structure, forcing it into low-energy configurations. Combaz and Neyret's system, like others, demonstrate that smooth organic forms typically emerge when physically modelling elastic material (Smith, 2006; Combaz and Neyret, 2002, 2006). Physicality plays a key role in the system introduced in the next chapter.

Representation A flexible representation of an organism is required in order to model complex 3D surfaces. A discrete boundary surface linear representation, such as a polygonal mesh, is conceptually simple, and, due to the arbitrary size and shape of the individual polygons, can approximate complex smooth surfaces with detail over multiple scales. As seen in this chapter, many developmental models operate on polygonal surfaces. Smith (2006) and Matela and Fletterick (1979) model cells as vertices, organisms as polygonal meshes, and developmental events as local mesh transformations. Cell division in these systems is modelled by replacing one vertex with two or more new vertices. Developmental events do not have to be modelled as local transformations. Kaandorp and Kübler (2001), for example, use a growth operator which operates on an entire polygonal mesh, modelling multiple biological processes simultaneously.

One key disadvantage of using a surface representation is that internal phenomenon of development can only be coarsely modelled, if at all. The mechanics of solid matter is one such phenomenon that can be only coarsely modelled using a surface representation. An example is given in a developmental model proposed by Smith, in which he uses an ad hoc "internal pressure" force to approximate the effects that internal forces have on the surface of a growing structure (Smith, 2006, Algorithm 9.3). This model simply applies a constant force to each vertex in a normal direction, and is only accurate for structures that have a consistent width — which is a major restriction on the set of forms that can be modelled. Modelling internal developmental events, such as cell division, using a surface representation can also only be approximately modelled, but are typically abstracted away. For example, sub-surface growth might be modelled using an increasing internal pressure force as described above, or by moving the surface cells outwards, giving the appearance of a growing volume. A volumetric representation can model both of these aspects of development at a higher degree of fidelity. The tetrahedral mesh, used in the system described next, is one such representation.

2.3.1 Conclusion

This chapter examined a range of existing generative systems that model biological and physical processes. These systems are capable of producing beautiful and complex 2D and 3D forms and patterns, and offer a glimpse into what is and will be possible in generative computer graphics. These systems partially fulfil the goals of this research, and it seems likely that a system capable of fulfilling them completely would incorporate the best features of these systems, which are outlined above. This thesis presents such a system, the Simplicial Developmental System, which is now introduced.

Chapter 3

The Simplicial Developmental System

This thesis presents the design, implementation and analysis of a new system, the *Simplicial Developmental System* (SDS), that supports the simulation of development and generation of complex organic geometry. SDS uses principles from biological development and physics to *grow* two and three dimensional organic forms from basic embryonic structures. It was designed to address some limitations of the systems discussed in the previous chapter and was inspired by work in theoretical biology (Matela and Fletterick, 1979), artificial embryology (Fleischer et al., 1995), and computer graphics (Prusinkiewicz and Lindenmayer, 1996; Smith, 2006; Kaandorp, 1994; Combaz and Neyret, 2006).

Broadly, SDS models a developing organism as a *simplicial complex* (a triangular or tetrahedral mesh) within a spatial environment. Cells of the organism move, grow, and divide, transforming the complex. Cells communicate with each other via a chemical signalling mechanism, allowing them to coordinate and build macro-level structures. SDS incorporates physical properties such as cell adhesion, elasticity and spatial constraints that result in the generation of organic form. Other spatial and environmental elements, such as tropisms, static meshes, or the presence of other developing organisms can be included in the simulation. The output of SDS consists of the *developmental sequence* of these forms, which is typically incorporated into an animation. Figure 3.1 shows an example simulation performed in 2D.

SDS is capable of generating organic volumetric forms with smooth surfaces. This is possible due to its discrete geometric representation and physical model of soft elastic material. It supports the emergence of geometric forms from many localised geometric transformations. This is demonstrated with the limb bud module (§5.1,§7.1)

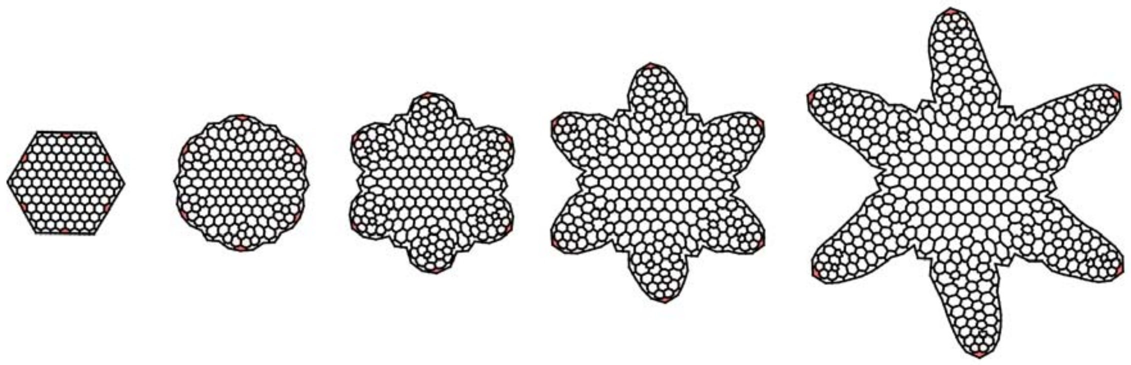


Figure 3.1: A sequence of frames from a 2D SDS simulation of a starfish-like form.

which grows tentacle-like protrusions through the orchestration of directed cell division, morphogen diffusion, and a feedback loop. The limb can be re-used within a model by implanting it in numerous locations, and organic variation amongst the limbs is achieved due to the environmental and geometric context-sensitivity of SDS. Figure 3.1, for example, was generating using the same cell "program" in each starfish arm, and yet each limb is subtly different. This natural repetition with variation is a key advantage of the SDS approach. The geometric modules in SDS are not strictly delineated within a form, rather, "natural" geometric interfaces emerge between module boundaries. In Figure 3.1, for example, there is no clear distinction between cells belonging to a starfish arm, and cells belonging to the body. This is a consequence of the emergence of modules from local geometric transformations, and avoids the discontinuous module boundaries evident in some module oriented 3D modelling systems.

This research was conducted in two main phases. Firstly, experiments with a 2D system examined how developmental and physical processes could be integrated. The second phase of the research looked at generalising principles from the 2D system to a 3D system. The commonality in both systems is captured in the general SDS framework for modelling biological and physical processes for organic form creation, the subject of this chapter. This thesis presents and compares some specific implementations of SDS in two and three dimensions, referred to as SDS2 and SDS3 respectively.

SDS is a complex system and it is useful to introduce each of its components separately: the simplicial complex representation of biological structure, the growth model, the model of cellular communication, the structural transformations, and the physical model. This chapter first presents an outline of these components and the role they play in the framework (§3.1). Each of the remaining sections then discuss each component in turn, starting with the component's biological relevance and history in existing systems, before presenting the details of the component. This

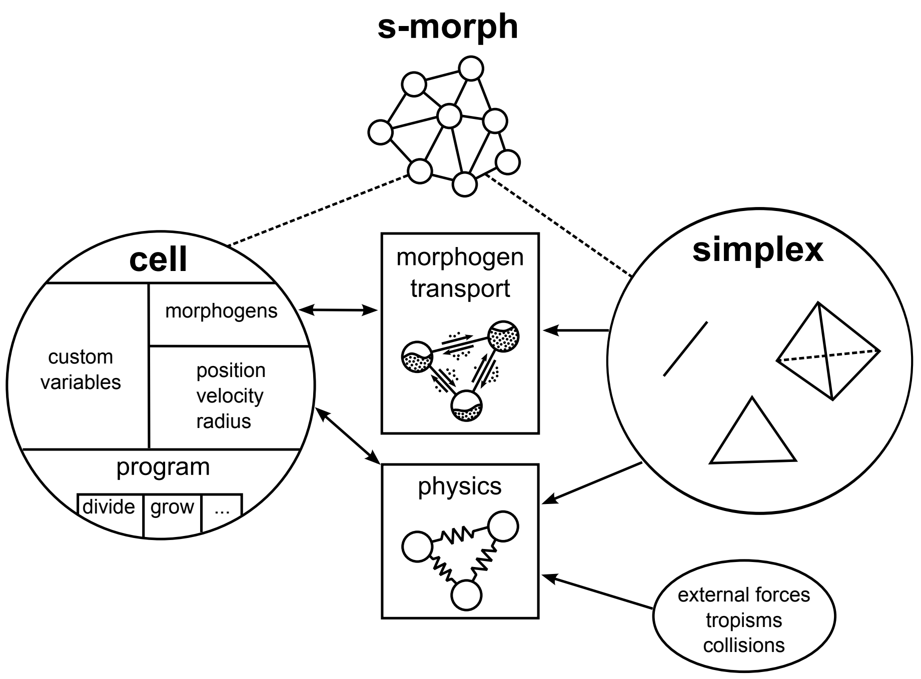


Figure 3.2: An overview of some of the key components of SDS. An s-morph is composed of cells and simplexes. Cells contain state (such as position) and execute a cell program that may perform actions (such as cell division). Morphogens diffuse between cells, and are transported along the edges of the simplexes. The physics component of SDS specifies the motion of cells by modelling simplexes as springs. External forces, collisions, and tropisms can also contribute to the motion of cells.

chapter introduces a number of new concepts and technical notation, a summary of which is presented in Table 3.2 in the concluding section of this chapter.

3.1 SDS Concepts

SDS is a simulation-based framework comprised of a number of components (Figure 3.2). This section introduces these components and explains how they fit together to create an overall simulation framework. The subsequent sections then explore each of the components in depth.

In SDS, a geometric form is generated by simulating morphogenetic and physical processes acting upon a structure termed a *Simplicial Morph*, or an *s-morph*. An s-morph consists of a set of *cells* — autonomous entities that execute actions based on internal state, received "information", and external stimuli. The cells of an s-morph are connected by the *structure* of an s-morph, which is composed of geometric elements known as *simplexes*. The structure of an s-morph is a triangular mesh in 2D and a tetrahedral mesh in 3D. Figure 3.3 illustrates an example three dimensional s-morph.

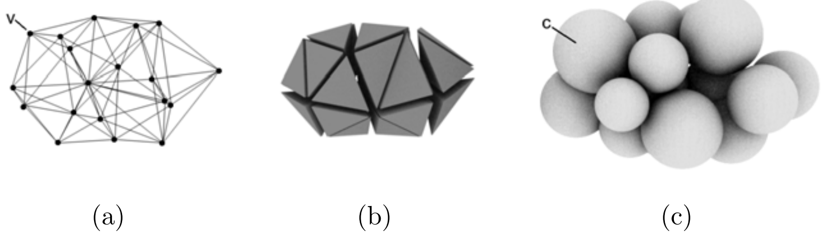


Figure 3.3: An example SDS3 s-morph. It has (a) vertices, edges and (b) tetrahedra. It also has (c) spherical cells. These are all different views of the same structure. Vertex v corresponds to the position of cell c.

Each cell in the s-morph operates as an individual agent or program. A cell can modify its properties, can distribute information to neighbouring cells, and can execute actions such as cell division. The behaviour of the cells is governed by the *growth model*, which includes the specification of a *cell program*, a set of morphogens, and a set of simulation parameters. The input to an SDS simulation includes an initial s-morph, a growth model and an environment. The growth model drives the development of the s-morph, which undergoes both continuous and discrete structural changes until the simulation is halted.

SDS allows cells to coordinate their behaviour using a simple morphogen model that is inspired by protein signalling in biological systems. Morphogens diffuse between adjacent cells and decay slowly over time. This simple mechanism allows cells to estimate distance from, and direction to, a morphogen source. This is used, for example, to determine the region of proliferation in a developing segment or limb (§5.1).

A cell in an s-morph may choose to divide, at which point it is replaced with two or more daughter cells. Adding or removing cells requires modifications to the structure of an s-morph. Maintaining a triangulation (or tetrahedralisation) of the structure poses many challenges, and thus cell division is not a simple operation. In addition to cell division, an s-morph's structure dynamically adapts when cells move. In SDS, this is known as a *structural movement transformation*. Division and structural movement are the two operations that change the structure of an s-morph, hence changing the developing form.

The motion of cells through space is described by a physical model, in which the structural elements of an s-morph (its edges and triangles or tetrahedra) are modelled as elastic springs. This has two purposes: firstly, cells can divide and grow and the surrounding structure will expand and re-organise to accommodate them, and secondly, the structure behaves and appears as if composed of soft elastic matter.

These components are incorporated into the *simulation loop*, outlined in Algorithm 1 (p35). The simulation loop iteratively "steps" the state of the world (the environment and the s-morph) forward in time by a time increment, Δt, specified by the user. One iteration of the loop involves several steps. Firstly, `SimulatePhysics` integrates the positions and velocities of all the cells in an s-morph over the specified time interval (§3.6). `MovementDetected` looks at the current state of an s-morph and detects whether a cell movement transformation needs to be performed (§3.5.3). If a cell movement transformation is required the simulation is rewound to the time of the first movement and the movement transformation is executed (`TimeOfFirstMovement`, `RewindSimulation`, and `PerformCellMove`). After this point, the world time may be advanced either by the full time step, Δt, or up to the rewound point, $\widetilde{\Delta t}$, therefore it is important that all modules take this into consideration. Collisions between surface elements of an s-morph are detected and handled by `HandleCollisions` (§3.6). `UpdateCellState` then updates other cell properties that can change over time, for example, cell radius (§3.6). After this, the flow of morphogens through the s-morph is simulated by `SimulateMorphogens` (§3.4). Finally, all the cell programs are executed by `RunCellPrograms` (§3.3). A cell program may instruct a cell to divide, at which point `PerformCellDivide` is executed (§3.5.2).

Algorithm 1 SDS Simulation Loop

```
$t = 0$
**while** $t < duration$ **do**
  SimulatePhysics($\Delta t$)
  **if** MovementDetected() **then**
    $\widetilde{\Delta t}$ = TimeOfFirstMovement()
    RewindSimulation($\widetilde{\Delta t}$)
    PerformCellMove($cell$)
  **else**
    $\widetilde{\Delta t} = \Delta t$
  **end if**
  HandleCollisions()
  UpdateCellState($\widetilde{\Delta t}$)
  SimulateMorphogens($\widetilde{\Delta t}$)
  RunCellPrograms($\widetilde{\Delta t}$)
  $t = t + \widetilde{\Delta t}$
  Output world state for time $t$
**end while**
```

Having presented an overview of SDS, each of its components are now discussed in detail. Discussion of the components are complemented by the examples in Figure 3.4, and it may be helpful to refer back to these examples while reading this chapter.

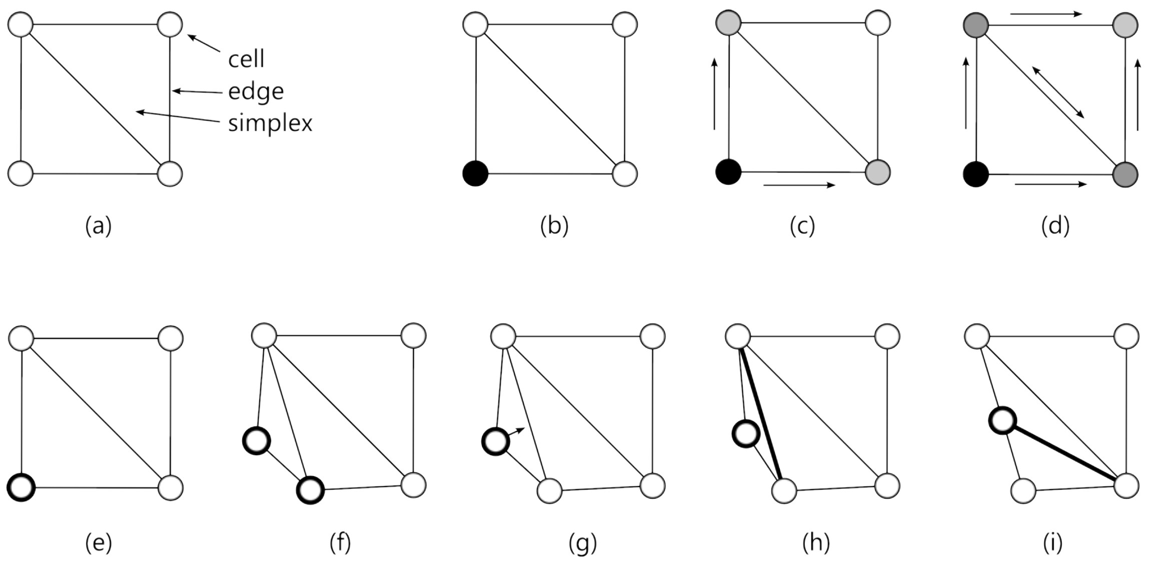


Figure 3.4: Key SDS components demonstrated in 2D. (a) *Representation.* The representation of a SDS2 s-morph consists of cells, edges and simplexes (triangles). This figure illustrates an example s-morph in 2D that has four cells. (b) *Morphogen diffusion in 2D.* A cell in a 2D s-morph starts to constantly produce a morphogen (the concentration of which is shown via the shading of the cells). (c) The morphogen diffusion equations cause morphogens to travel between connected cells from high concentration to low concentration, therefore the two neighbours of the black cell start to fill up with morphogen. (d) After a number of time-steps the morphogen is distributed throughout the s-morph. (e) *Cell division in 2D.* The highlighted cell in this s-morph has elected to divide. (f) Upon division the cell splits into two new cells, and the s-morph is adapted to maintain a simplicial complex structure. (g) *Cell movement in 2D.* Cells may move through space in reaction to physical forces. In this example the highlighted cell is being pushed in the direction shown by the arrow. (h) Occasionally cells may come into contact with the edges of the s-morph, as is occurring in this example. (i) As the cell crosses the edge, the cell movement operation restructures the mesh to maintain non-overlapping simplexes. In 2D this is achieved by simply "flipping" the highlighted edge.

The most important component of SDS, the representation of a developing biological structure, is discussed first.

3.2 Representation

There are a number of ways a developmental system can be represented: symbolically, geometrically, topologically, mathematically, and so on. A defining characteristic of SDS is its representation. SDS models a developing organism as a set of cells bound together with a mesh. The primary goal of this research was to generate complex 3D surfaces for use alongside 3D modelling packages, and hence the output of SDS is oriented towards producing geometric models in a suitable form. In SDS3 a user can extract the boundary of the s-morph's tetrahedral mesh to obtain a triangulated surface mesh — a geometric representation compatible with most 3D modelling packages. This section discusses some potential geometric representations for developmental modelling, introduces the geometric representation used in SDS, and discusses the relationship between an s-morph and its geometric structure.

3.2.1 Background

Numerous theoretical and computational models of shape exist, including interpolating representations, implicit and explicit surfaces, iterated function systems, and constructive solid geometry. This section outlines some different representations and motivates the representation scheme used in SDS.

Interpolating representations are primarily applied to modelling surfaces. In this representation, a shape is typically defined using a set of *control points* in space and a method for generating a surface by interpolating between them. This approach is useful for generating smooth surfaces[1] and is the basis of many commercial modelling applications. Whilst these representations can be extremely sophisticated, it is not obvious how to use this representation within the developmental paradigm, and to the author's knowledge, no such system exists.

The functional paradigm includes all the methods of representing shape using mathematical functions. Functional representations include implicit surfaces, iterated function systems, and parameterised surfaces (see e.g., (Bloomenthal and Bajaj, 1997),(Flake, 1999, §7), and (Schneider and Eberly, 2003, §9.7)). Implicit and parameterised surfaces are the most suitable of these for modelling smooth form. Implicit surfaces (or volumes) define shapes as a set of points where a specified function evaluates as zero (or negative). For example, a sphere of unit radius can be defined using the equation $x^2 + y^2 + z^2 = 1$. The equation is true for all points (x, y, z) that lie on the sphere. Practicality requires that implicit functions be constructed using high-level components, such as skeletal shapes (Bloomenthal, 1995). Parameterised, or *explicit*, surfaces are defined using images of functions. A simple example is the two-dimensional parameterised line segment, $f(t) = p(1 - t) + qt$, which maps the parameter space segment $[0, 1]$ to a line joining two points, p and q.

In general, functional representations are concise and are capable of producing smooth, organic shapes, and complement specific modelling domains, such as particle-based fluid modelling (e.g., metaball-based (Blinn, 1982)). The functional representation seems suitable as a scheme that will ultimately be controlled by a developmental system; although discrete models have more advantages for complex modelling.

[1]There are numerous interpretations of *smoothness* in computer graphics (e.g., C^1-continuity); however, in this context the common use of the term is accepted.

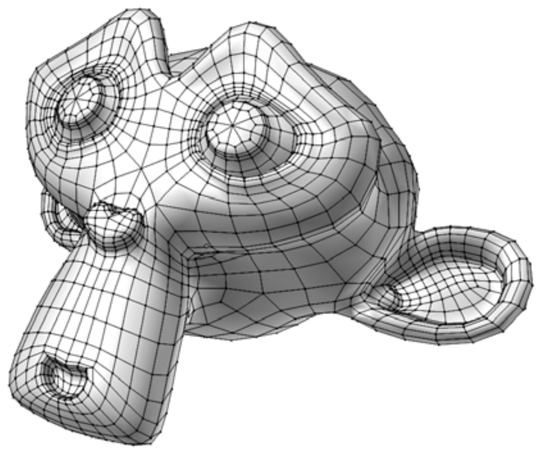


Figure 3.5: A polygonal surface mesh consists of vertices, edges and polygons.

The representation of shape by collections of discrete elements is a popular strategy in computer graphics and is the dominant representation used in developmental modelling. Developmental modelling typically breaks a biological system into many parts or modules, and so it makes sense to use a discretised geometric representation.

Constructive Solid Geometry (CSG) combines discrete elements such as cubes, spheres, and cylinders using Boolean operations such as union, intersection and difference. The method is typically applied to represent solid sharp-edged forms in the engineering industry; however, there are extensions that make the method suitable for modelling smoother organic forms (Barthe et al., 2004).

For 3D modelling the polygonal surface representation is by far the most popular. The polygonal surface mesh models an object using vertices, edges, and polygonal faces (see Figure 3.5). This representation is popular in domains such as computer games, character design for animated films, and CAD, and most modern 3D modelling packages use this representation (e.g., Blender[2]). Modern graphics hardware accelerates the rendering of this representation and numerous creative applications exist that directly support this model. There are myriad methodologies and algorithms that support this representation. One such operation helpful in organic modelling is *subdivision surfaces* (see e.g., (Zorin and Schröder, 2000)) which refines and smoothes a polygonal mesh.

Vertex-Vertex systems (§2.2.3), Kaandorp and Kübler's growth model (§2.2.2), and the semi-interactive morphogenesis system (§2.2.4) all represent organisms with polygonal meshes and demonstrate different ways to model growth and development on the mesh. These systems are revisited later (§6.1).

[2]Blender is an open-source 3D modelling and animation suite (`http://www.blender.org/`).

Modelling only the surface of a developing form has some limitations which can be solved by using a volumetric representation. Typical volumetric representations of shape use collections of discrete solid elements such as cubes, hexahedra or tetrahedra. Modelling a volumetric form as a collection of axis-aligned same-sized cubes is referred to as the *voxel* representation (§2.1.3). This representation is simple, fast and has been used successfully in developmental systems (e.g., Greene, 1989). A major disadvantage of the voxel representation is that, due to the uniform size of its elements, it is computationally expensive to represent objects that have both large and small scale detail. This can be addressed by spatial subdivision schemes, such as octrees, or by using fast GPU representations and algorithms (e.g., Crassin et al., 2009).

Tetrahedral meshes are a highly flexible volumetric representation historically used within finite element modelling in computational engineering. Techniques for real-time physical simulation of tetrahedral meshes have recently popularised this representation in computer graphics (e.g., Teschner et al., 2004). The 2D analogue of the tetrahedral mesh — the triangular mesh — was originally proposed as a representation of cellular layers by Matela and Fletterick (1979). The model was developed primarily because it provided greater flexibility than the cellular automata approaches common at the time. §4.1.3 reviews their model in more detail. Triangular meshes in 2D and tetrahedral meshes in 3D provide an elegance and simplicity to the conceptualisation and implementation of developmental models. With respect to the research goals outlined in the first chapter these representations have numerous benefits, including:

- conceptual simplicity,
- generality (they work in 2D, 3D, or indeed any dimension),
- they are supported by existing physical simulation techniques (§3.6),
- flexibility (they can model complex topologies and detail over multiple scales),
- compatibility with 3D modelling tools (allowing integration into such tools), and
- allow modelling of interior as well as surface detail.

These advantages were the primary motivation for the choice of the triangular mesh in 2D and the tetrahedral mesh in 3D as the representation scheme for the structure of s-morphs.

3.2.2 The S-morph

An *s-morph* in SDS consists of a set of cells, C, and a mesh with edges, E, triangular faces, F, and, in SDS3, tetrahedra, T.

In SDS2 an s-morph consists of a set of cells connected together with a triangular mesh, and in SDS3 the cells are connected together with a tetrahedral mesh. Figure 3.3 shows an SDS3 s-morph. Triangular meshes in 2D and tetrahedral meshes in 3D can be unified using the concept of *simplicial complexes.* A simplicial complex[3] in n dimensions is a topological structure that is composed of a set of simplexes, where a *simplex* is an edge, face, or tetrahedron (called a 1-simplex, 2-simplex, and 3-simplex respectively). The simplexes join a set of vertices in space, and overlapping and intersecting simplexes are not allowed. Stated using this terminology, an s-morph in SDS consists of a set of connected cells and a simplicial complex. This abstraction helps to simplify comparisons between SDS2 and SDS3.

The SDS2 representation has a heritage in the 2D triangulated graph representation of cell layers introduced by Matela et al (see §2.1.4); however, simplicial complexes have not been used before in a 3D developmental system. A volumetric representation has the advantage of being able to model internal developmental and physical processes, allowing an extra level of expressiveness not possible in surface-oriented systems. For example, in the limb bud model presented later (§5.1) it is the proliferation of *internal cells* that causes outward growth of limb-like forms.

A *cell* is an axiomatic entity in an s-morph. It can perform actions, such as divide, and has a set of properties which may change over time. These include cell position, c_x, velocity, c_v, and radius, c_r (see Table 3.2 for a list of all the cell properties.) A cell is modelled as a circle in SDS2 and as a sphere in SDS3, with a specified radius, c_r. The density of all cells is uniform and so $c_m = \text{vol}(c)$, where the volume, $\text{vol}(c)$, is computed as:

$$\text{vol}(c) = \pi {c_r}^2 \qquad \text{(SDS2)} \qquad (3.1)$$

$$\text{vol}(c) = \frac{4\pi}{3} {c_r}^3 \qquad \text{(SDS3)} \qquad (3.2)$$

Two cells are *neighbours* if there is an edge connecting them — this neighbourhood relationship forms the *topology* of the s-morph. Neighbouring cells diffuse morphogens between one another and can detect each other's properties (such as morphogen concentration). The function $N : C \to 2^C$ maps a cell to the set of its neighbours.

[3]The terminology of simplicial complexes differs between fields, but see e.g., Popović and Hoppe (1997) for a complete formal definition.

The cells of an s-morph can transform its simplicial complex. When a cell moves, the adjacent simplexes change shape. If the movement of a cell causes a simplex to collapse, the local structure is modified to prevent simplexes from overlapping (§3.5.3). If a cell divides then the structure also changes (§3.5.2).

There are some important restrictions on an s-morph's mesh. Firstly, the mesh elements cannot intersect. This supports the embeddedness criterion specified in the previous chapter. This constraint can be satisfied by detecting and preventing collisions between elements. In SDS3 this is done by incorporating collision detection and handling into the physical simulator (§6.3.5). The second restriction is that *hinges* are invalid (see Figure 4.6e). A triangle in SDS2 or tetrahedron in SDS3 represents a solid *chunk* of matter, and so a hinge represents an infinitesimally thin joint within an s-morph. For physical and biological realism, as well as simplicity, an s-morph is considered to be solid everywhere, and consequently hinges are not allowed.

An s-morph can be viewed from two perspectives. From the cell's perspective, an s-morph is made up of cells that grow, divide, and move. The cells execute a program that drives the generative process, and it is through the program that a user has primary control of the system (§3.3). From this perspective, the simplexes are merely byproducts of the topology of the cells. The alternative is the simplex perspective which focuses on the structure of the s-morph on which the physical and morphogen models act. Both perspectives are useful in understanding the development of an s-morph. In actuality, the cells and simplicial complex are tightly coupled: the simplicial complex defines the physical equations (§3.6) which describe the motion of the cells, and the motion and behaviour of the cells affect the structure of the simplicial complex through structural transformations.

The representations of s-morphs are revisited in §4.2 and §6.2, when details specific to the 2D and 3D implementations are discussed. The next key element of SDS discussed is its model of cell behaviour.

3.3 Growth Model

In SDS, cells behave as autonomous entities, absorbing and diffusing morphogens, changing their internal state, and modifying the s-morph through cell division and growth. The description of the behavior of the cells and other factors that lead to the generation of specific forms are grouped in SDS under the concept of *growth model*. This section briefly reviews some related research into existing models of cell behaviour before presenting in detail the approach used in SDS.

3.3.1 Related Work

The universal abstraction in developmental systems is that of the *autonomous cell*. There are many different models of cellular autonomy, all abstracting the role of the genome and the effects of proteins in biological systems. These models range from abstract models of cells as computers, to rich biologically inspired dynamical models.

Some systems model the cell as a computer that executes logical instructions in sequence and communicates via formal protocols. The Cell Programming Language (CPL) (Agarwal, 1994) is a simulation system built primarily to support experimental observation. A discretised CA space[4] is populated with cells that execute a cell program. The user defines the genomic and environmental influences on the cell behaviour using a simple language which offers high-level constructs like `divide`, `move`, `for_each_neighbour_do`, `die`, conditional and jump instructions, and biochemical control commands. The cell program is executed at each time step in parallel in all of the cells.

An abstraction that is closer to biology than the cell computer approach is the Genetic Regulatory Network (GRN). A cell is modelled with a finite set of genes that are active or inactive. Active genes can cause the production of a protein, activate or repress other genes, and can be activated or repressed by protein thresholds. The coupling between genes and proteins is then described by a network. More abstract forms of these networks (Random Boolean Networks, or RBNs) have been identified to contain interesting dynamics (see e.g., Kauffman, 1995). Such a network is used in Dellaert and Beer's system, in which they explored the evolution of a developmental model for autonomous agents with a simple morphology and control system (Dellaert and Beer, 1994). Their model is divided into the organism, cell, and biochemical levels. At the lowest level cell DNA is modelled as a bit vector and genes are modelled as individual bits that are either *on* or *off*. The dynamics of the DNA is modeled with a RBN. Each cell contains a bit vector indicating which genes are active. This is updated at each time step using the current state and the state of its neighbours (iterated repeatedly until the network enters a stable state.) The cell divides if a certain gene is activated. Other mechanisms, like communication and differentiation are also included. A key result of Dellaert and Beer's research is that the evolution of the agents effectively reinforced and extended a developmental pathway (in the RBN) that eventually built successful agents.

[4]CPL is designed to be applicable to any cell space representation (e.g., 2D grid, 3D mesh, linear), however for his experiments Agarwal has explicitly used a 2d hexagonal-lattice with cells represented by a set of connected points.

A less abstract model is provided by Streichert *et al.* in which they present a complex system that grows groups of cells (Streichert et al., 2003). They evolve the organisms to be limited in their growth (i.e., to grow to a certain size and stop) and to self–repair if damaged. Their model incorporates continuous space, is structure–oriented, and has dynamic neighbourhood relations between cells. Cell behaviour is controlled by Random Boolean Networks (RBNs) and S-systems[5] They also implement endogenous communication between cells. Their model incorporates a simple physics in which cells have a uniform size and attempt to attach to their seven closest neighbours. Adhesion forces the cells into stable configurations.

L-systems implement autonomy through rules that execute based on context, cell type and state. Given a cell with a particular type, state, and context, the grammar encodes a replacement rule that acts on that cell. There are a finite set of rules and types, however (parameterised) state may be continuous.

The *connectionist* approach of Mjolsness, Sharp and Reinitz provides a phenomenological framework for modelling development (Mjolsness et al., 1991; Krul et al., 2003). Their approach abstracts collections of interacting cells, proteins and genes. It integrates continuous dynamics with grammar-based rules of cellular actions. The general idea is to construct a matrix of continuous values that defines the interaction between all pairs of genes, where a positive value indicates activation, a negative value indicates repression, and a zero value indicates that no interaction occurs. Certain assumptions about the system (e.g., gene effects are additive) lead to a set of simplified differential equations describing the dynamics of the system. This model was used in Fleischer's system to model and evolve primitive neural networks and shapes out of cells roaming on a plane (§2.1.5).

3.3.2 Cell Programs in SDS

In SDS, a *growth model* determines the kinds of form that are generated. It requires the implementation of a cell program, the specification of a set of morphogens, and the definition of cell and simplex variables. Cell programs are implemented with a module `RunCellPrograms(`Δt`)` that is called at each time step of the simulation. For the results presented in this thesis the cell programs are modelled using a set of rules of the form:

$$r_i : \text{condition} \rightarrow \text{action}$$

[5]S-systems model a dynamical system as an abstract set of values related by a non-linear differential equation of a specific form. They have been used, amongst other things, to model genetic regulatory networks in theoretical biology (Irvine, 1988).

The *condition* is a Boolean expression containing terms such as morphogen values, cell properties, and the state of the cell's neighbours. The condition is checked every iteration of the simulation loop, and if it is satisfied then the action is executed. The *action* may be a structural action, such as the cell division operation, divide(d), which instructs the current cell to divide in direction d. Alternatively, it may be a change in cell state, for instance allowing a cell to change radius, c_r, or create morphogen. As an example, consider the rules in Table 3.1. Rule r_1 specifies that upon becoming half full of morphogen ϕ, a cell should divide towards the source of ϕ. The $\nabla\phi$ term models the ability of a cell to detect its local morphogen gradient. Rule r_2 commands all surface cells to produce morphogen ϕ_1 at a rate K. The final rule, r_3, specifies that all the neighbours of a cell of type A should increase their radius at a linear rate of 0.01 units per second. A variety of other rules are demonstrated in Chapters 5 and 7.

Table 3.1: An example cell behaviour rule.

rule	condition	action
r_1	$c_\phi > \frac{1}{2}\,\mathrm{vol}(c)$	$\mathrm{divide}(\nabla\phi)$
r_2	$c_{surface}$	$c_{\Delta\phi_1} = K$
r_3	$c_t = A$	$\forall n \in N(c) : n_{\Delta r} = 0.01$

Rule-based cell programs are concise and suited to the results shown in this thesis; however, at the lowest level cell programs are implemented in code, and hence arbitrary computations within a cell can be performed. Extensions to this research which could lead to more user-friendly interfaces are considered in §9.6. Biological cells do not act in isolation, and it is vital, if macro-level structures are to be generated, that the actions of cells be coordinated. The next section presents the model of cell communication used in SDS to coordinate cell actions.

3.4 Cell Communication

Cells in SDS are modelled as autonomous entities in a simplicial complex, each operating according to a set of supplied rules (§3.3.2). In order to generate structures in SDS which are composed of many cells, it is important that cell actions be coordinated. The principle mechanism used to coordinate cells in SDS is a model of cell communication inspired by chemical signalling in biological systems. This section first discusses how cell communication and coordination are modelled in existing developmental systems, and then presents the model used by SDS.

3.4.1 Background

The successful growth of an organism relies heavily on the ability of cells to organise and coordinate their actions. This requires cells to be able to communicate. Biological cells communicate via proteins and receptors which bind to their surfaces. Once bound, a receptor causes the release of another protein, internal to the cell, which results in a series of reactions that may activate or repress a gene, thus causing a change in the cell[6]. The coordination and organisation of the developing embryo and complex organs, such as the vertebrate eye, rely extensively on protein signalling (Gilbert, 2006, p143).

General communication and coordination amongst cells is vital in biological systems, and many artificial developmental systems incorporate some form of information sharing. In L-systems, information can be acquired locally through context. The context-sensitive rule "$a < b > c \to d$", for example, specifies that b changes to d if it has an a on its left and a c on its right. Context-sensitive rules can be used to model the flow of information or nutrients within a structure. As an example, consider the repeated application of the rules "$m < a \to m$" and "$m \to a$" to the structure *maaaa*. The developmental sequence that results is: *maaaa*, *amaaa*, *aamaa*, *aaama*, *aaaam*; which can be interpreted as a message, m, propagating through the structure. This technique has been used to propagate a flower-activating signal in a model of plant development (Prusinkiewicz and Lindenmayer, 1996, p32).

The coordination of development is not constrained solely to explicit messaging between cells. Structures can also be shaped by other forms of "information" including, for example, the presence of light, walls, or food. Open L-systems (Měch and Prusinkiewicz, 1996) introduced a "special symbol reserved for bilateral communication with the environment". This allows environmental information to be passed into the growth rules, for example the amount of light available at a leaf could affect its growth rate. The use of environmental information to guide the growth of a structure is demonstrated effectively by Greene's Voxel Automata 3D models of vines growing across walls and competing for light (Greene, 1989). Other systems, such as Kaandorp's model, show that cohesive structures can emerge when diffusing food within an environment (§2.2.2). In these systems information that exists in the environment helps to coordinate the developmental processes.

Alan Turing postulated in 1952 that structural and temporal interactions of proteins (he referred to them as *morphogens*) generate complex dynamics in biological systems (Turing, 1952). Systems similar in spirit to Turing's original models have been

[6]Signalling can also occur by passing proteins through special *gap junctions* in the cell membrane of adjacent cells

proposed in theoretical biology (Meinhardt, 1982) and computer graphics (Turk, 1991), and produce beautiful and complex spatio-temporal patterns (see §2.2.5).

Reaction-diffusion systems are interesting and generate very complex patterns; however, the research of this thesis explores very basic patterns and their effect on growing forms. One simple pattern, for example, is the gradient of morphogens that emerges by continuously producing and diffusing a decaying morphogen from a cell or region. In biological systems, these gradients can be used by cells to infer coordinate systems or positional information (Wolpert, 1969). Early experiments with the limb bud model (§5.1) showed that SDS didn't need a very complex communication mechanism (at least at this early stage), all it needed was a mechanism for producing morphogen gradients. This model is described next.

3.4.2 A Morphogen Model of Communication

In SDS, a model of cell communication based on morphogen patterning has been chosen. The cells in an s-morph contain morphogens that diffuse between neighbouring cells and decay over time. This model allows cells to communicate over short distances and for morphogen gradients to be established. The module `SimulateMorphogens`$(\widetilde{\Delta t})$ simulates the flow of morphogens around an s-morph over a time interval. The flow of morphogens around an s-morph could be modelled in a number of ways, some of which are illustrated in Figure 3.6. SDS uses the model shown in Figure 3.6b as it is the simplest approach capable of modelling morphogen gradients. The model can be summarised as follows:

- There is a finite set of morphogens,
- Morphogens exist *inside* cells where they are measured with a concentration value,
- Morphogens can be created or destroyed within a cell,
- Morphogens are transported around the s-morph by diffusion,
- Morphogens decay over time,
- Every cell accepts and diffuses *all* morphogens,
- Cell membranes are negligible,
- Signalling is juxtacrine[7] (occurring only between adjacent cells), and

[7]Cell signalling in early development can be categorised as *juxtacrine* or *paracrine*. Juxtacrine signalling occurs between two neighbouring cells by passing proteins through gap junctions in the shared membrane, and paracrine signalling involves diffusing proteins through the extracellular matrix (over short distances).

- Diffusion is isotropic.

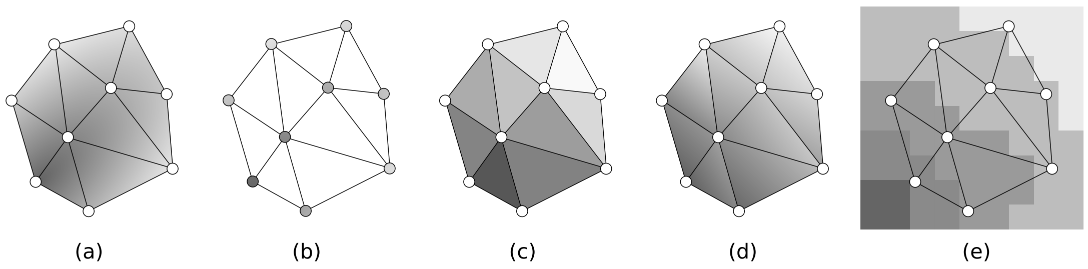


Figure 3.6: Different morphogen models in 2D. The shading represents morphogen concentration in the system. (a) The "ideal" model in which morphogen diffuses continuously throughout the shape. This would require an analytic representation, which only exists for trivial distributions. Some alternative models include: (b) restricting the morphogen to exist only within cells, modelling the morphogen as (c) constant or (d) linearly varying within each triangle, or (e) discretising the space and modelling morphogens as constant within each spatial unit.

A *morphogen* is a cell variable, denoted as c_ϕ. The morphogen concentration is a continuous value in the range $[0, \mathrm{vol}(c)]$. Each morphogen has a rate of diffusion, D_ϕ, and a rate of decay, C_ϕ, that is specified in the growth model. Diffusion of morphogens occurs between neighbouring cells and morphogens decay within cells. This is modelled using the standard particle diffusion equation:

$$\frac{\partial \phi}{\partial t} = D_\phi \nabla^2 \phi - C_\phi \phi \tag{3.3}$$

This equation is discretised over the s-morph structure following the assumptions given above, and is simplified by the fact that morphogens are represented as a single value within cells and diffuse isotropically. The specific discretisation for SDS2 and SDS3 is presented later (§4.6,§6.6).

At this point the structural representation, cell behaviour model, and cell communication models used in SDS have been presented. The cell behaviour model lets cells perform actions. Some of these actions, such as cell division, causes new cells to be added to the s-morph. This requires that the simplicial complex be adjusted to *make room* for the new cell. This cell division operation, and the structural operation of cell movement, are now discussed.

3.5 Structural Transformations

In developmental systems a *structural transformation* is a discrete modification of the structure of an organism or form. For example, adding or removing a symbol in an L-system string, splitting a cell wall in a Map L-system, or collapsing an edge in Vertex-Vertex systems are all structural transformations.

In SDS, structural transformations modify the structure of an s-morph by adding or removing cells, edges and simplexes. Two transformations have been explored in this research: cell division and cell movement. Cell division is a transformation that replaces one mother cell with two or more daughter cells. Structural cell movement (not to be confused with the normal spatial movement of a cell) is a transformation that dynamically adapts the structure of an s-morph as cells move across simplex boundaries. Structural transformations can furthermore be categorised either as active or passive. Cell division is an active transformation as cells can choose to execute it. Structural cell movement on the other hand, is automatically performed when the right conditions occur, and as such, is passive.

This section briefly reviews and rationalises structural transformations in related systems and then provides an overview of the two transformations used in SDS.

3.5.1 Background

In nature, early biological development consists of processes such as: cleavage divisions, pattern formation, morphogenesis, cellular differentiation and growth (see e.g., Gilbert, 2006). This thesis is primarily concerned with *morphogenesis*, the process by which cells proliferate, organise and form complex structures. The fundamental processes of morphogenesis are (Gilbert, 2006, p13):

- cell division,
- cell movement,
- cell death,
- cell growth,
- cell shape changes, and
- changes in the composition of cells and secreted products.

Cell division, or *mitosis*, involves a complex process of genome duplication and membrane cleavage (Gilbert, 2006, p111). Through this process the multitude of cells that constitute a multicellular organism are generated. Another important process in morphogenesis is the movement of cells in an organism or developing embryo. Cell movement supports many fundamental processes: the aggregation of dispersed cells; the relocation of groups of cells; the dispersal of locally manufactured cells around an organism; and the formation of connective cell networks (Davies, 2005, p96).

It has been extensively demonstrated in simulation that cell movement (through differential adhesion) can self-organise specific structures and patterns (Matela and Fletterick, 1979; Hogeweg, 2003). The death of a cell can arise necrotically from poisoning, membrane rupture, physical stress and starvation. Cell death can also be a programmed part of development where it is referred to as *apoptosis* (Gilbert, 2006, pp158–160). This can be advantageous to an organism for redistributing resources or when creating complex structures, such as in the formation of mammalian fingers or toes via death of inter-digital tissue (Gilbert, 2006, pp522–523).

In SDS, the *structural transformations* are cell division and cell movement. The other processes are not considered as structural transformations and are modelled in the physical model (cell growth) and growth and morphogen models (changes in the composition of cells and secreted products). Cell death is not considered in SDS, as it was never required in the growth models designed; however, cell death could broaden the range of forms the system can generate, and could be easily accommodated within the current framework (§9.2.1).

Structural transformations such as cell division, growth, and movement, have been modelled in different ways. Arguably the most elegant formalisation of cell division is given by L-systems, where the simple rule $a \rightarrow bc$ signifies that a cell of type a will divide into two cells of type b and c. Moreover, their relative positions are given: b is to the "left" of c. Other systems incorporate cell division or *part replacement* depending on the structure they use. Other L-system transformations include: $a \rightarrow b$ which replaces a geometric part with another, $ab \rightarrow ba$ which moves parts around, and $ab \rightarrow a$ which deletes a part.

Structural transformations are essential in developmental systems and the complexity of performing transformations depends on the structural representation used. In L-systems, for example, modelling the division of a single cell is a trivial operation; however, in Map L-systems, it involves a number of steps. Map L-systems are L-systems that operate on planar maps, or equivalently, graphs (Lindenmayer and Rozenberg, 1979). In this representation, the edges of the graph model cell walls, and the faces or regions model the cells. The axiomatic operations in this system add, replace, delete or subdivide cell wall segments. To perform cell division, i.e., dividing a face in two, requires a rigid sequence of cell wall manipulations. In SDS, particularly in the 3D case, the division and movement transformations are very complex. A primary contribution of this thesis is the exploration of these transformations and comparison of some different approaches.

Other systems model structural transformations more abstractly. For example, in Kaandorp's model of accretive growth (described in §2.2.2) the single structural transformation is a “growth step”, in which a new mesh layer is created around the old one. For each vertex of the mesh, a growth function determines the direction and amount of local growth that occurs. The mesh is adapted as necessary to preserve or remove detail. In this model, as with the other continuum models, cell division is abstracted away to surface growth.

Later in this thesis, details on how these and other developmental systems perform structural transformations will be further discussed (§4.1,§6.1.2). We now consider how cell division in SDS can be modelled.

3.5.2 Cell Division

Cell division in SDS involves the transformation of an s-morph by replacing a dividing *mother* cell with two or more *daughter* cells. From a structural point of view, the division transformation needs to remove the mother cell from the simplicial complex, add the new daughter cells and reconfigure the local structure so that no simplexes overlap and no hinges occur. Cell division in SDS is performed by the module `PerformCellDivide`, outlined in Algorithm 2. `PerformStructuralDivision` is the key component, it transforms the structure of the s-morph and returns the set of daughter cells, C'. After the structure has been transformed, the mass of the mother cell is distributed evenly amongst the daughter cells. The growth model may need access to the newly created cells (to distribute morphogens or to assign special cell variables), and so `PerformCellDivide` returns the daughter cells.

Algorithm 2 `PerformCellDivide(c,d)`

```
PerformStructuralDivision(c,d)
Let $C'$ be the newly created cells
Remove $c$ from $C$
for $c' \in C'$ do
    $c'_m = \frac{c_m}{|C'|}$
    compute $c'_r$ from $c'_m$
end for
Return $C'$
```

Structural cell division can be implemented in a number of ways, ranging from subdividing a neighbouring triangle in SDS2 (Algorithm 6 (p78)) to Delaunay tetrahedralisation schemes (Algorithm 11 (p127)). A key contribution of this research is the presentation and comparison of different approaches (§4.4,§6.4). A range of factors must be considered when implementing a cell division algorithm, for example: should the neighbourhood connections of the mother cell be split in exactly two amongst the daughter cells?

Assuming that a cell divides in a direction d, into exactly two daughter cells, $C' = \{a, b\}$, experimentation with SDS has shown that the following considerations are important when implementing a good structural cell division algorithm:

1. a and b should be neighbours (localised),
2. $b_x \approx a_x + d\alpha$, for some $\alpha \in \mathbb{R}$ (directional division), and
3. $|N(a)| \approx |N(b)|$ (topologically symmetric division).

The first rule states that the daughter cells should be neighbours. This corresponds to a physical understanding of how cell division in nature occurs. In addition, a structural cell division operation should only modify the local structure. If the effect of a dividing cell propagates throughout an s-morph it may be difficult to localise development and form coherent modules. The effects could either be structural modifications that propagate throughout a mesh or a big "jump" in the physical model[8]. The issue of local transformation is discussed later (§8.5).

The second rule states that the daughter cells should lie in the specified axis of division. The degree to which the approximation holds is dependent on the division technique. For example, simplex subdivision in SDS2 (Algorithm 6 (p78)) only loosely approximates this equation, whereas the balanced division algorithm (Algorithm 4) satisfies it.

The third rule evenly distributes the mother's topological connections amongst the daughter cells. Maintaining balance this way across an s-morph makes the system easier to use, and was required for the forms generated in the experiments. There may be other requirements of a cell division transformation, including physical stability and geometric efficiency (discussed later §8.1).

From a generative perspective, cell division is about adding new elements to an s-morph, and hence a division operation is not limited to producing just two daughter cells. Algorithm 4 (p73), for example, uses additional *stabilising* cells in order to distribute the neighbours more symmetrically between the daughter cells. Sections 4.4 and 6.4 present some different algorithms for cell division in triangular and tetrahedral meshes. The second important structural transformation in SDS, that of cell movement, is now considered.

[8]A jump, or discontinuity, in the physical equations corresponds to a dynamical catastrophe (Thom, 1975).

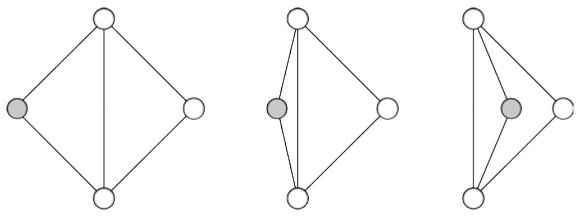


Figure 3.7: (left to right) A cell (shaded) in SDS2 crosses over an edge, causing the two triangles to overlap.

3.5.3 Cell Movement

The cell movement transformation allows the structure of an s-morph to adapt to the movements of cells. As cells move through space they may move into positions where simplexes overlap (see Figure 3.7), which is not permitted. One solution would be to naïvely prohibit simplexes from overlapping, by detecting and preventing collisions between cells and simplexes. In SDS, this technique is used for boundary cells that collide with boundary simplexes (see §3.6). For all the other cases, SDS *reconfigures* the mesh, giving cells more freedom in moving around within an s-morph. This approach balances the extremes of having a very rigid structure (e.g., a mesh that doesn't change at all) and a very loose structure (e.g., Fleischer's loosely coupled cell system (Fleischer, 1995)).

The early experiments performed with SDS2 showed that this technique leads to visually organic arrangements of cells (Figure 3.1). This adaptive operation also leads to low energy structures, which is highly beneficial when using integration methods that may become unstable under high stresses.

Cell movement is implemented in SDS by a number of modules. `MovementDetected`() looks at the current state of an s-morph and detects whether a cell movement transformation is required. During a single time-step, many cell movement transformations may need to be performed. While it could be possible to perform multiple structural transformations in parallel, a simpler approach is to rewind time back to the first detected movement, and then perform a single structural movement. `TimeOfFirstMovement`() and `RewindSimulation`($\widetilde{\Delta t}$) carry out these two tasks. In the implementation of SDS2, it was assumed that — as the time-step was very small — only one cell movement ever occurred at a time, so these modules were only implemented in the 3D system. The `PerformCellMove`(*cell*) module performs the cell movement operation. By assuming that only one cell movement has occurred, the algorithm is greatly simplified.

Cell movement in 2D can be performed with a trivial edge-flip operation (§4.5). This operation does not generalise to 3D, and as a result, the 3D transformation is significantly more complicated (§6.5).

Active movement Cell movement can be considered as an active transformation, for example in modelling cell migration. Cell migration is an important part of biological development (Gilbert, 2006) but was not extensively considered in this research. Models of cell-directed movement could be incorporated into SDS either by allowing cells to control their motion, or by adding explicit movement actions (for example, a transformation that swaps the positions of two adjacent cells).

The two key transformations presented here, cell division and cell movement, allow an s-morph to grow in complexity and let cells distribute themselves throughout an s-morph. The cell movement transformation discussed here is triggered as a response to cells moving around spatially, pushed around due to the forces modelled in the s-morph. The physical model that describes this motion is now discussed.

3.6 Physical Model

A key goal of the research presented in this thesis was to model the deformations that occur when soft objects come into contact. These deformations are modelled in SDS through the use of a physical model, implemented as a physical simulation that runs in parallel to the structural transformations, morphogen simulation and growth model. In a nutshell, s-morphs are modelled as mass-spring systems, with the cells as point masses and the simplexes as springs. The simulation causes the cells to move around, forcing the s-morph into a shape that has low potential energy. Unlike related developmental systems, SDS3 uses a deformable model of local volume conservation. The system handles collisions amongst surfaces, which means that the components of an s-morph never intersect in space. The physical model of matter in SDS has the following properties:

- a growing region of the s-morph forces its surroundings to expand and re-organise around it,
- regions of the s-morph resist compression, giving *solidity* to an s-morph, and
- the surface of an s-morph doesn't intersect either with itself or with other objects in the world.

This section first looks at how physical models have been used in computer graphics and developmental systems, and then presents in detail the approach used by SDS.

3.6.1 Background

Many physical factors influence biological development, including gravity, pressure, surface tension, temperature, radiation, magnetic fields, friction, and viscosity. Evolution has exploited these physical laws to generate the various functional forms we find in nature. D'Arcy Thompson emphasised the importance of studying these physical effects alongside traditional biology in order to truly understand biological development (Thompson, 1942). Many patterns and forms we find in nature can be explained by simple processes governed by physical laws, including the arrangements of bubbles in foam, zebra stripes, sand dunes, and mineral dendrites. (These examples and more can be found in Ball's *The Self-Made Tapestry* (Ball, 2001) which explains many different models of pattern and form development.) In cellular systems, simple processes can lead to complex organic structures. Hogeweg, for example, demonstrated that a variety of natural patterns such as segmentation and budding could emerge through differential cell adhesion alone (Hogeweg, 2003).

Computer graphics uses physical models to achieve visual realism and realistic dynamics. In modern computer games, for example, physical simulation middle-ware is often incorporated to provide many realistic physical effects, including collisions between objects, fluid dynamics, and rag-doll physics. Physical simulation is also used in 3D animation and modelling to model phenomenon such as ocean waves, fracturing objects, and fire and smoke effects (O'Brien and Hodgins, 1999; Fournier et al., 1987). In developmental modelling, the incorporation of a physical model is an effective way to achieve natural forms, as has been noted by computer graphics and artificial life researchers alike (Eggenberger, 2003; Jirasek et al., 2000; Kaandorp and Kübler, 2001). Combaz and Neyret's system (§2.2.4) most elegantly demonstrates that a simple growth process coupled with an elastic physical model results in complex organic folded structures. Details of some physical models used in developmental systems are discussed later (§4.1, §6.1). Section 6.1 also reviews a number of different approaches to physical modelling in computer graphics in general.

3.6.2 Modelling Physics in SDS

The physical model in SDS was originally designed to address the following problem. Consider the s-morph in Figure 3.8. An edge between two cells indicates that they are neighbours and should be in contact, hence the length of an edge should equal the sum of the radii of the cells it connects. Figure 3.8 (a) shows an ideal situation in which the cells are arranged so that neighbours are touching. As the center (shaded) cell grows it would be ideal to move all the cells into a configuration such that all neighbours are touching but not overlapping. Image (b) illustrates that, in this example, this isn't possible, as the gaps between the cells cannot be closed sufficiently. This dilemma could be resolved in a number of ways: For example, by allowing the cells to take arbitrary polygonal shapes, or by changing the topology by removing the edges that are too long or too short.

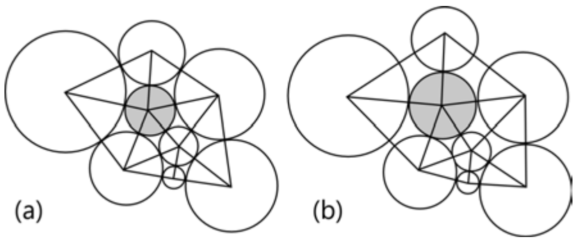


Figure 3.8: Deformation of an s-morph due to cell growth. (a) An s-morph contains cells and a mesh (both shown). (b) As the shaded cell changes its size, the mesh deforms. Ideally we would like all neighbouring cells to touch sides, but as this example demonstrates, this is generally impossible, and thus this requirement must be relaxed.

SDS follows a physical modelling approach in which an energy term measures how close an s-morph is to an ideal configuration. The goal of the simulation algorithm is then to reduce the energy of an s-morph to a minimum. The strategy adopted by SDS is simple and works well, in 2D a mass-spring system is used and in 3D a generalised mass-spring system is used, in which both edge and tetrahedral springs. This approach provides a simple approximation of the complex dynamics within a soft body and is common in physical simulation (Müller et al., 2008; Turini et al., 2007). Mass-spring systems have also been used previously to model the physics within cell complexes (Eggenberger, 2003; Streichert et al., 2003; Smith, 2006; Lindenmayer and Rozenberg, 1979).

Re-using the same mesh throughout the different aspects of SDS (physics, cell communication, and geometry) keeps SDS conceptually simple and relatively easy to implement. Other physical simulation methods, such as mesh-free simulation (reviewed later in Section 6.1.1) would arguably provide a more sophisticated and accurate simulation, but this would incur a higher computational cost and significantly increase the complexity of the implementation. Moreover, the results shown in this thesis demonstrate that the mass-spring approach is sufficiently capable of producing a variety of organic shapes efficiently, and with perceptually believable physical behaviour.

The mass-spring model in SDS defines energy-minimising forces on an s-morph's mesh which causes the geometry to behave as a soft elastic object. In 2D this is achieved by modelling the edges as springs, and in 3D by modelling both the edges and tetrahedra as springs (Figure 3.9a). For each simplex, s (edge, face, and tetrahedron), the dynamics of the simplex spring is controlled with a stiffness

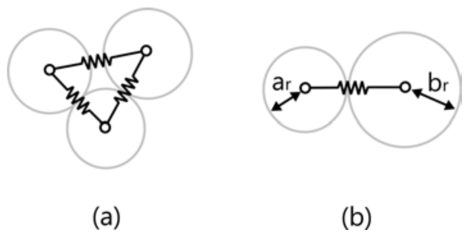


Figure 3.9: Calculating the rest length of an edge. (a) A simple SDS2 s-morph with three cells connected with edge springs. (b) The rest length of an edge connecting two cells a and b is determined to be the sum of their radii: $a_r + b_r$.

coefficient, s_{k_d}, and damping co-efficient, $s_{k_{damp}}$. The radii of the cells, c_r, specify the desired rest state of all the edges, faces, and tetrahedra (Figure 3.9b). Neighbouring cells wish for their surfaces to be just "kissing" and resist attempts at pulling them apart or pushing them together. To inhibit the intersection of the surface of an s-morph, a collision detection and handling system was incorporated (though this was only required and implemented in SDS3).

Biological cell growth combined with mitosis results in a massive increase in size of an organism. Different rates of growth, or *allometry*, are evident in developing organisms. Some developmental systems cannot model cell growth due to limitations in the representation of structure, cellular automata being a prime example. The simplicial complex representation used in SDS does not suffer from this drawback.

The physical simulator is incorporated into SDS via the `SimulatePhysics`(Δt) module which updates the physical properties of an s-morph by the specified time increment. The simulator also provides the `HandleCollisions`() module, which detects and handles collisions within the system. This includes collisions within an s-morph that occur between its boundary cells and boundary simplexes, collisions between s-morphs and static geometries within the world, and collisions between multiple s-morphs. The final component of the simulator is `UpdateCellState`($\widetilde{\Delta t}$), which updates other cell variables that have a continuous rate of change. For example cell radius, c_r, is modified by the rate of change variable $c_{\Delta r}$. Theoretically this rate of change is continuous, $c_{\Delta r}(t) = \frac{dr}{dt}$, but due to the discrete, step-wise nature of the simulation, $c_{\Delta r}$ is assumed to be constant over each time interval $[t, t + \Delta t]$. With relatively small step sizes this simplification is unlikely to affect the modelling capability of SDS, and could always be improved if finer control is required.

The physical model is an important component of SDS. By modelling simplexes as elastic elements, an s-morph always has an appearance of being soft, squishy, and organic. Detecting and resolving collisions between boundary cells and simplexes ensures that the modules of an s-morph never intersect and spatially interact in a natural manner.

3.7 Conclusion

This chapter introduced the Simplicial Developmental System. SDS defines a representation called the s-morph (§3.2) and a simulation framework for iteratively transforming the s-morph. The system is composed of a number of components: the physics simulator (§3.6), the cell behaviour model (§3.3), the morphogen simulator (§3.4), and the structural transformations (§3.5). The SDS framework states at a general level how each of these components should work, the remainder of the thesis details exact implementations in 2D and 3D.

When presenting his developmental system, Fleischer postulated the question: "Why should we have any faith that this abstraction will work?" (Fleischer, 1995, p14). The goals of SDS are creative rather than scientific, and if "work" is taken to mean "able to generate interesting and complex forms", the question still applies. By modelling some of the fundamental processes of morphogenesis (§3.5.1) we can begin to have confidence that the system is capable of reproducing some interesting developmental phenomena. SDS incorporates a number of these processes, including cell division and cell movement, but not others, such as cell death. These processes were chosen to support the implementation of some biological models of development (§5) and, as this thesis demonstrates, are sufficient to generate complex organic geometries. This level of abstraction was sufficient for the goals of this research, but there is no doubt that the incorporation of other developmental processes will increase the expressiveness of SDS and the range of forms it can generate (see §9.2). There are many more interesting theoretical questions about developmental systems — one particularly interesting example is, "What is the minimal set of biological processes necessary to generate structure X?" These kinds of questions may be addressed in future research, but lie outside the scope of the research presented here.

The remainder of this thesis discusses how SDS can be implemented in two and three dimensions, compares alternative designs of SDS components, and most importantly, demonstrates that SDS can successfully generate complex organic forms. A large number of symbols were introduced in this chapter that will be used throughout this thesis. Table 3.2 summarises this notation for reference. The next chapter describes how SDS can be implemented in two dimensions.

Notation	Description
c	cell
c_x	cell position
c_v	cell velocity
c_m	cell mass
c_r	cell radius
$c_{\Delta r}$	rate of change of cell radius
c_{ϕ_i}	concentration of morphogen ϕ_i in cell
$c_{\Delta\phi_i}$	rate of change of morphogen ϕ_i in cell
$c_{surface}$	true if a cell lies on the boundary of an s-morph
$c_{\dots}$	custom cell variable (e.g., c_{frozen} (§6.3.8))
$N(c)$	set of neighbours of cell c
$\mathrm{vol}(c)$	volume of cell
s	a simplex (edge, face, or tetrahedron)
s_{k_d}	stiffness coefficient of s
$s_{k_{damp}}$	damping coefficient of s
$s_{\dots}$	custom simplex variable[a]
$R(s)$	rest size of simplex s
$V(s)$	actual size of simplex s[b]
$E(s)$	energy of a simplex
$F_s(c)$	force acting on cell c by simplex s
$F(c)$	total force acting on c
Φ	set of morphogens for a growth model
ϕ_i	morphogen i
D_{ϕ_i}	morphogen diffusion coefficient
C_{ϕ_i}	morphogen decay coefficient
$\nabla\phi_i$	gradient of ϕ_i
$\nabla^2\phi_i$	Laplacian of ϕ_i
C	set of cells of an s-morph
E	set of edges of an s-morph
F	set of faces of an s-morph
T	set of tetrahedra of an s-morph
S	set of all simplexes of an s-morph
$c \in s$	true if simplex s is attached to cell c

[a] An arbitrary variable can be associated with a simplex (e.g., the simplex rest scale multiplier in §6.3.7).

[b] The actual size of a simplex can be negative (see Equation 6.10).

Table 3.2: Notation used in this thesis.

Chapter 4

A 2D Developmental Modelling System

This chapter presents the details of a 2D implementation of SDS, called SDS2. This system was initially prototyped in order to design and explore the concepts behind a novel physically simulated and embedded developmental system. The eventual goal was to build a generative 3D form system, with the view that much of the design of SDS2 could be easily generalised into 3D. Two biologically-inspired models of development were implemented using SDS2: a limb bud model (used to generate the form shown in Figure 4.1) and a stripe generation model. These models were used in some experiments performed with SDS2 and demonstrate how SDS is able to generate forms and patterns. These results are presented in the next chapter. This chapter describes the design of SDS2, and includes an examination of:

- different algorithms for modelling cell division on a triangular mesh,
- how structural cell movement reduces to a simple edge-flip operation, and
- the physical coupling between cell system and triangular mesh.

Many existing developmental systems operate in two dimensional space, some of which were outlined earlier (§2.1.4,§2.1.5). SDS2 draws on a number of features from existing systems. It has a structural representation similar to Matela and Fletterick's triangulated graph model (§4.1.3), a rule-based model of cellular behaviour similar to Fleischer's multi-mechanism model (§4.1.2), and incorporates a physical model acting upon a structure (as found in Cell Systems (§4.1.1)). SDS2 unifies these features into a common framework and incorporates new functionality, such as allowing cells to passively modify the structure of an s-morph as they move, and allows cells to communicate by modelling morphogen flow. Before the details of

SDS are discussed, it is important to consider how previous related systems model development in 2D.

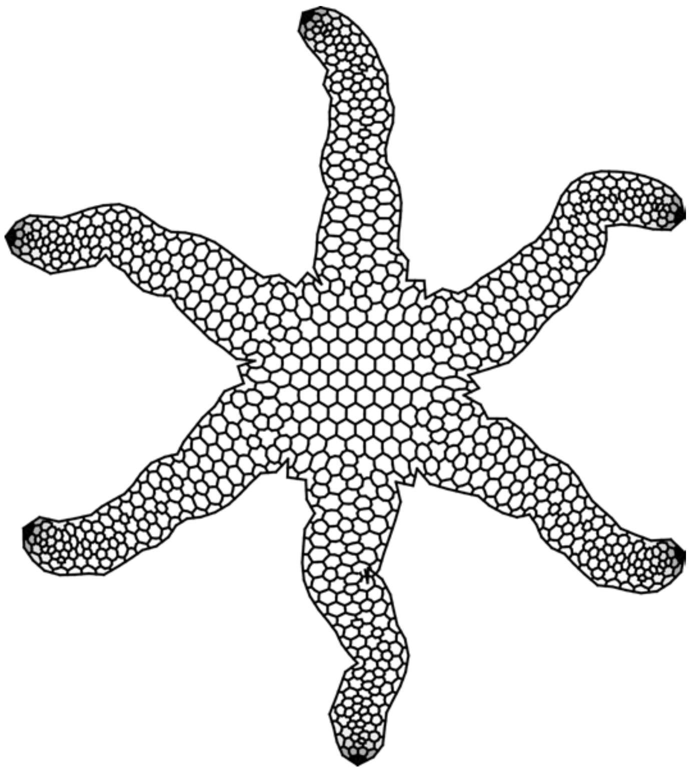


Figure 4.1: A starfish–like form generated using SDS.

4.1 Related Work

This section examines some existing methods for modelling development in 2D and provides context for the design of SDS2. Comprehensive reviews, found in Stanley and Miikkulainen (2003) and Prusinkiewicz (1993), can be consulted for descriptions of other 2D developmental models as this section only describes the work most closely related to SDS.

4.1.1 Map L-Systems and Cell Systems

Map L-systems generalise L-systems to operate on 2D models of cell structures (Lindenmayer and Rozenberg, 1979). Cell layers are represented as graphs, where (labelled) edges represent cell walls, closed faces represent cells and rewrite rules operate on the edges of the graph. A developmental model consists of a labelled graph and a set of rewrite rules. A single developmental step transforms the graph using the rewrite rules. Expressing the developmental model as a set of rewrite rules greatly simplifies working with a 2D structure. However, some operations, such as cell division, require a complex sequence of wall rewriting rules.

The complexity of modelling cell operations in Map L-systems was addressed by the *Cell Systems* approach, which focusses on cell operations, rather than wall operations (de Boer et al., 1992). Cell Systems supports the simulation of 2D pattern and shape formation for exploring morphogenesis in the context of theoretical biology. It incorporates a physical model of mechanical cell interactions caused by osmotic pressure and wall tension. Additionally it incorporates vector fields (e.g., gravity, morphogenetic fields) in order to provide cells with directional information they can use when dividing.

A cellular structure in a Cell System consists of a finite set of cells, each with a polygonal shape and state information. The cells are non-overlapping polygons that form a polygonal mesh by connecting together. The physical model (called a *pressure-tension* model) consists of two components: cells exerting pressure on their boundary walls, and walls modelled as linear springs with masses at the vertices of the cell polygons. The cellular structure changes upon cell division, which is controlled by L-system-like production rules. The rule "$A \to B \uparrow (\alpha)C$", for instance, indicates that a cell of type A divides into two cells of type B and C with the division wall oriented at an angle of α degrees from the reference vector at the center of A. The development proceeds as a sequence of steps, where each step involves applying all production rules, computing the steady state of the physics system, and updating the vector field.

Map L-systems and Cell Systems show how a grammar-based approach of development can be applied to generate 2D structures. Although Cell Systems support arbitrary polygonal cell shape and model material physics, it differs from SDS2 in a number of aspects. The physical model in SDS2 models connections between cells with springs, whereas in Cell Systems, cell walls are modelled with springs. Cells in SDS2 are free-floating and can move around within the adaptive mesh, whereas in Cell Systems, cells are fixed to a specific neighbourhood. Unlike Cell Systems, SDS2 also incorporates a method of cell communication, allowing cells to coordinate their activity over short ranges. These differences primarily exist because SDS2 was explicitly designed to be generalisable to 3D. It is not obvious how a system that models cells as polygons (like Cell Systems) can be easily generalised to 3D, without being overly complex. Nonetheless, Map L-systems and Cell Systems are elegant techniques for modelling the development of 2D cell systems.

4.1.2 Fleischer's Model

At the opposite end of the spectrum to L-system methods lies Fleischer's multi-mechanism model. This model considers cells as autonomous circular entities that are continuously simulated in a 2D plane (§2.1.5). It is illuminating to examine some of the details of this system. The fundamental entity is a cell, which, like SDS, has a behaviour and a set of properties. Specifically, a cell has:

- *state*: that models properties and intracellular and surface chemicals,
- *sensors*: that are mapped into the cell state equations,
- *state equations*: differential equations that model changes in state, and
- *behaviour functions*: which map cell state to actions and state changes.

An example behaviour function is (from Fleischer, 1995, Eq2.8):

$$\text{TimeToSplit}(\mathit{state}, \mathit{env}) \equiv \mathit{env}[\mathit{radius}] \tilde{>} r_0 \ \widetilde{\text{and}}\ \mathit{state}[\mathit{split}] \tilde{>} \mathit{split\_threshold} \tag{4.1}$$

This equation indicates that a cell should split when its radius exceeds r_0 and when it has accumulated enough of the *split* protein. The '$\tilde{>}$' and '$\widetilde{\text{and}}$' operators are continuous analogues used in place of the Boolean $>$ and `and` operators, due to the continuous nature of the system.

The model incorporates physical aspects of cell systems, such as viscous drag acting on individual cells, mechanical barriers, collisions between cells, and cell adhesion (Fleischer, 1995, p13). The circular cells move around the environment governed by high viscosity dynamics $F = mv$ with a viscous drag, $k_{drag} = \frac{32}{3}\eta r$, where η is the fluid viscosity. Cells have a motive force which can be specified via their behaviour function. It can also be bound directly to their state variables, for example (Fleischer, 1995, Eq2.6):

$$\text{MotiveForce}(\mathit{state}) \equiv (\mathit{state}[fx], \mathit{state}[fy]) \tag{4.2}$$

A model of cell adhesion allows cells to stick together, where the adhesion forces are computed from the contact area between the cells and the concentration of surface factors (binding chemicals). The collision force between cells is computed based on their overlap using a collision detection mechanism that also allows collisions with other objects in the environment. The collision and adhesion forces are combined using an empirically derived function.

The system is simulated by first gathering all the different components into a large system of piecewise ODEs (PODEs). This system is then numerically solved up

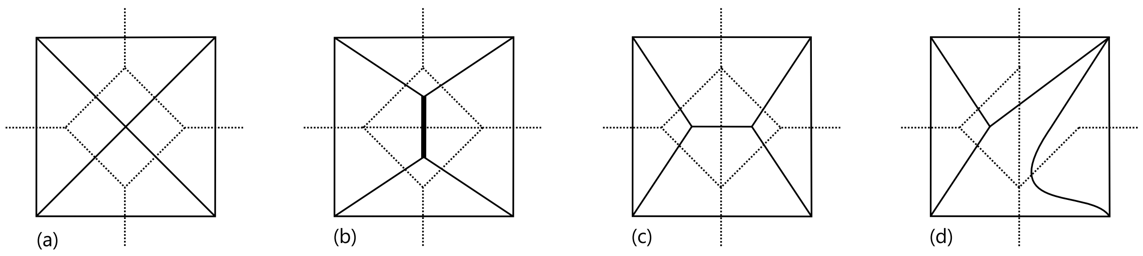


Figure 4.2: (a) A cellular layer modelled as a planar map, and its topology (dotted) with a graph. This sequence of illustrations demonstrates the effects of the primitive operations on the topology and the conceptual cell map. (b) The insertion of an edge (bold) results in a change on the topology, but not in the number of cells. (c) An edge exchange is performed, changing the topology of the layer. (d) An edge deletion drastically changes the topology.

until a discrete event occurs, such as cell division. When such an event occurs, the simulation is halted, the relevant action performed, and then the simulation is resumed.

Fleischer's system is different from SDS in that everything is expressed as PODEs which are solved. SDS describes some components using ODEs (the physical model and the morphogen model), however the focus is on incrementally updating the world state, rather than as a system of equations to be solved. These two approaches offer different ways to specify cell behaviour, either as a set of equations, or as a program. The difference is purely conceptual as the two are theoretically interchangeable (a program could solve a set of equations and a set of equations could describe most cell programs).

4.1.3 Matela and Fletterick's Model

Matela and Fletterick's model of cellular layers was introduced earlier (§2.1.4). The original formulation used planar maps to represent cellular layers, where the regions correspond to cells. The graph represents the dual of the cell map, and under this interpretation cell layers may be considered as planar graphs where cells are nodes, and edges indicate neighbourhood relationships. In the original paper (Matela and Fletterick, 1979) three basic operations were proposed: insertion, deletion and the exchange of edges (demonstrated in Figure 4.2).

Matela and Fletterick showed that modelling a cellular layer with a general graph results in biologically unstable networks, and so they restricted their attention to the triangulated graph, which "was chosen because observation of living tissues shows this pattern to be most common" (Matela et al., 1983, p360). From a structural point of view the triangulated graph reduces the set of primitive edge operations to just one: the *edge exchange*, since other operations would void the triangulation.

Exchanging edges in a graph models the movement of cells. Using this operation and two cell types, it was shown that cell self-sorting can occur under different conditions (Matela and Fletterick, 1980). Edge exchange, or the *edge flip*, is an important operation because for any set of points with a triangulation, any other triangulation can be obtained through a sequence of edge flips (Ransom and Matela, 1984, p237). This means that the edge-flip is sufficient for modelling cell movement at a topological level.

Matela and Fletterick's original model was later extended to model cell division (Matela et al., 1983; Ransom and Matela, 1984) and cell death (Duvdevani-Bar and Segel, 1988). Matela et al's model of cell division on triangulated graphs (Matela et al., 1983; Ransom and Matela, 1984) is important to this discussion because the representation of cell complexes is similar to that of SDS, and the cell division operation is governed by rules similar to the SDS division guidelines (§3.5.2). These rules are (from Ransom and Matela, 1984, pp238–240):

1. division involves the formation of a new vertex,
2. the new vertex is positioned on the graph adjacent to the dividing vertex,
3. changes in the pattern of edges around these two vertices are made to re-establish the triangularity condition,
4. the number of neighbours around the new and dividing vertices are balanced as equally as possible, and
5. (implicitly) cells divide into progeny which are both of equal size.

The cell division operation that implements these rules is based on a series of *division masks* (see Figure 4.3). The masks enumerate each of the configurations in which the dividing vertex has between four and nine neighbours and provide transformed subgraphs for each of these. Ransom and Matela noticed that this division mechanism often resulted in cells with very large numbers of neighbours, so they provided a balancing mechanism that allowed cells with more than nine neighbours to locally rearrange themselves through edge exchanges (Matela et al., 1983). Figure 4.4 shows a cell layer transformed with an edge exchange, followed by a cell division

The systems reviewed in this section illustrate different methodologies to modelling developmental systems in two dimensions. SDS2 has aspects in common with these systems, and can be considered as a synthesis of both Matela and Fletterick's model, with its representation of a cell system with a triangulated mesh and division operation, and Fleischer's model, with independent autonomous cells which are governed by rules and communicate via morphogen signalling. A significant feature of SDS2 that has not been previously incorporated in existing systems (to the author's knowledge) is the use of an adaptive triangular mesh to allow more flexible cell movement (§4.5). Having reviewed related work, SDS2 is now presented.

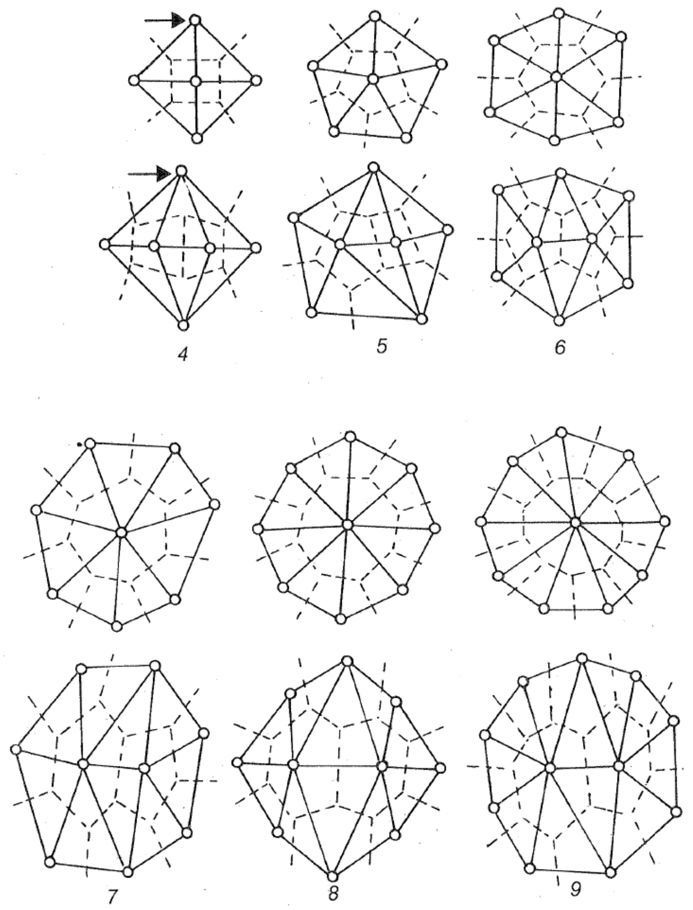


Figure 4.3: The *division masks* used in Matela et al's model. Each pair of illustrations shows the structure before and after cell division, for cells with between four and nine neighbours (from Ransom and Matela, 1984, Figure 4). Image courtesy of Raymond Matela.

4.2 SDS2 Overview

The system presented here was created as a precursor to a 3D modelling system, and as such, design decisions were oriented towards facilitating the transition to 3D. This included aspects such as the representation of an s-morph as a triangular mesh, the design of the structural transformations, and the re-use of the s-morph structure in the physical and communication models. Ultimately, the design of SDS3 was

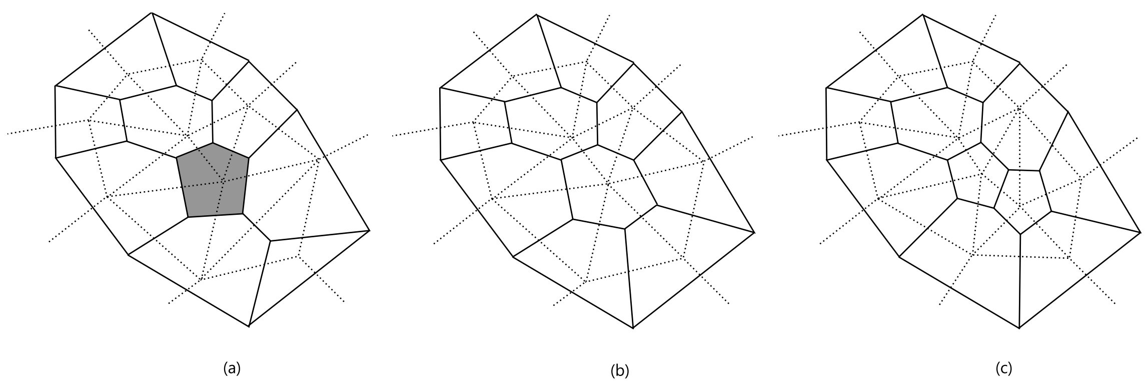


Figure 4.4: (a) A cell complex with a cell (shaded) migrating towards the lower right. (b) The movement is modelled as an edge exchange, and now the migrating cell is in contact with the lower right cell. (c) The migrating cell divides into two, resulting in the new complex shown.

formed by generalising the components of SDS2. Conceptually, this generalisation is straightforward — triangles become tetrahedra and circular cells become spherical cells. Technically, however, the generalisation posed many challenges (see Chapter 6).

The 2D prototype implements a subset of the full framework, including only the components necessary to demonstrate the effectiveness of the technique. The prototype omits collision detection and doesn't rewind the simulation for cell movement transformations. This is primarily because SDS was formative when the prototype was built, and it was found that collision detection can be omitted if there are no surface-surface collisions. Rewinding the simulation for the cell movement transformation guarantees certain structural conditions (see §6.5); however, if the time step is small enough, then only one cell movement will occur per step and rewinding is unnecessary. Algorithm 3 covers the main simulation logic, incorporating the major components of this implementation. The components are the s-morph itself, the physics model, the cell division and movement transformations, and the morphogen diffusion model.

An s-morph in SDS2 consists of a set of cells in 2D space connected together with a triangulated mesh. Figure 4.5 illustrates an example s-morph and some visualisation methods. Examples of valid and invalid s-morphs are shown in Figure 4.6.

The major components of SDS2 discussed in this chapter are:

- The physics model in SDS2, which models an s-morph as a mass-spring system (§4.3),
- A comparison of cell division algorithms (§4.4),

Algorithm 3 SDS2 Simulation Loop

```
$t = 0$
while $t < duration$ do
  SimulatePhysics($\Delta t$)
  if MovementDetected() then
    PerformCellMove($c$)
  end if
  UpdateCellState($\Delta t$)
  SimulateMorphogens($\Delta t$)
  RunCellPrograms($\Delta t$) (may call PerformCellDivide)
  $t = t + \Delta t$
  Output world state for time $t$
end while
```

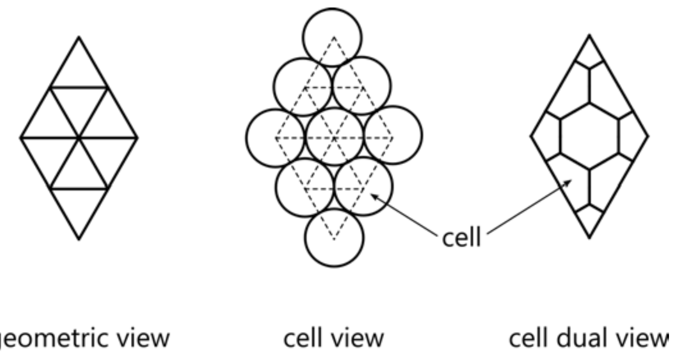


Figure 4.5: An SDS2 s-morph visualised in three ways. The geometric, or mesh view, shows the simplexes of the s-morph; in this view the cells are shown as vertices. The cell view shows the cells as circles, which is useful for visualising the sizes of the cells. The *cell dual view* visualises the cells and provides a much clearer illustration of an s-morph. In the dual view, a cell, c, is represented as a polygon, with $|N(c)|$ sides (one for each neighbour) and neighbouring cells share polygon edges. This view corresponds roughly to the graphic dual of the mesh graph; however, the sizes of the polygons reflect the sizes of the cells (as is also demonstrated in Figure 4.8).

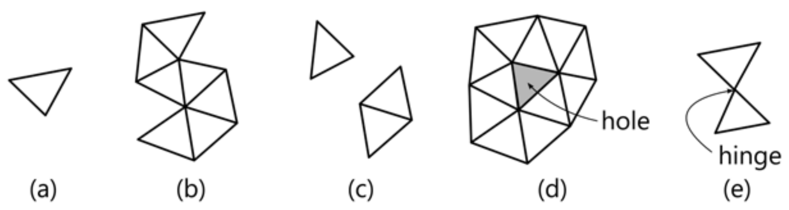


Figure 4.6: The meshes of some s-morphs. (a) The simplest 2D s-morph consists of three cells (represented by vertices in the mesh), three edges and a single triangle. (b) More complex s-morphs are composed of sets of joined triangles. (c) An invalid disconnected s-morph. (d) Another s-morph that is invalid due to a hole inside it. (e) An invalid s-morph due to the presence of a hinge.

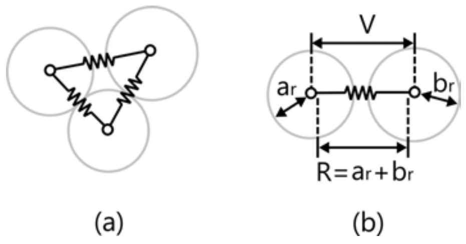


Figure 4.7: Edge springs in SDS2. (a) Three adjacent cells and the edge springs connecting them. (b) The rest length, R, of an edge spring is the sum of the radii of the two cells it joins. The actual length, V, is the distance between the two cells. The edge spring acts to minimise the difference between V and R.

- A model of cell movement, implemented as a simple edge-flip operation (§4.5), and
- The discretisation of the morphogen model over a triangular mesh (§4.6).

4.3 Physical Model

S-morphs in SDS2 are modelled using mass-spring systems, in which cells are modelled as point masses and edges are modelled as springs. The purpose of the springs is to preserve the local structure of the geometry. The system is solved using a standard real-time numerical integration scheme (the mid-point method) with damping added to increase stability. Collision detection and response can be incorporated into the simulator to prevent the triangles of an s-morph from intersecting; however, the developed prototype did not employ collision detection, as it was not found to be necessary to achieve the results shown in Chapter 5.

The physical simulation system in SDS2 implements the `SimulatePhysics`(Δt) routine in the simulation loop. The model treats cells as point masses with mass $c_m = \pi {c_r}^2$ connected together by springs along the edges[1]. The edges have a desired rest size, $R(s)$, which they attempt to maintain by applying forces on the adjacent cells. These elements are all connected via dynamical equations, which are then numerically integrated.

The dynamical equations The dynamical equations specify the motion of the cells by measuring the energy of an s-morph and forcing it into a lower energy

[1]The primary difference between this physical model and the model in SDS3 (§6.3) is that the 3D model uses additional tetrahedral springs, in order to preserve local volume. The analogue of this in 2D is to constrain the area within the triangles. This feature was not implemented in the SDS2 prototype as the concepts in SDS were still formative; however, it is likely that adding these triangle springs to an s-morph would increase its structural stability and allow the more regular cell arrangements.

configuration. A *potential energy*, $E(s)$, for each edge spring is computed from its current length, $V(s)$, rest length, $R(s)$, and stiffness coefficient, s_k, using Hooke's Law (Equation 4.3). The stiffness coefficient is spring dependent.[2]

$$E(s) = \frac{s_{k_d}}{2}\left(V(s) - R(s)\right)^2 \tag{4.3}$$

For a spring connecting two cells a and b (and remembering that c_x and c_r denote the cell's position and radius) we have $V(s) = |a_x - b_x|$ and $R(s) = a_r + b_r$ (this is shown in Figure 4.7). After computing the current and rest lengths the potential energy can be computed. The energy of each spring, s, generates a force on each adjacent cell that acts to minimise $E(s)$:

$$F_s(c) = -\frac{\partial E(s)}{\partial c_x} \tag{4.4}$$

This equation can be understood by considering a spring, s, connecting two cells, a and b. Taking $s_k = 1$, the derivative of the potential energy of the spring with respect to a_x gives the spring force acting on cell, a, as:

$$F_s(a) = -\frac{\partial E(s)}{\partial a_x} = (1 - \frac{R(s)}{V(s)})(b_x - a_x) \tag{4.5}$$

Note that if $R(s) = V(s)$, then the edge is in its desired state, and the force is zero. If $R(s) > V(s)$ then the edge is too short, and a is pushed away from b. Finally, if $R(s) < V(s)$ then the edge is too long, and a is forced towards b. The total force, $F(c)$, acting on a cell is the sum of forces from all the adjacent springs. This summed force affects the cell's position through the second order ODE:

$$\frac{d^2 c_x}{dt^2} = \frac{F(c)}{c_m} \tag{4.6}$$

Splitting the equation into two first order ODEs, and applying the equation over the simulation interval $[t_0, t_0 + \Delta t]$ gives:

$$c_x(t_0 + \Delta t) = c_x(t_0) + \int_{t_0}^{t_0+\Delta t} c_v(t)dt \tag{4.7}$$

$$c_v(t_0 + \Delta t) = c_v(t_0) + \frac{1}{c_m}\int_{t_0}^{t_0+\Delta t} F(c)dt \tag{4.8}$$

[2]For the experiments shown in the next chapter the stiffness coefficient is generally the same for all springs.

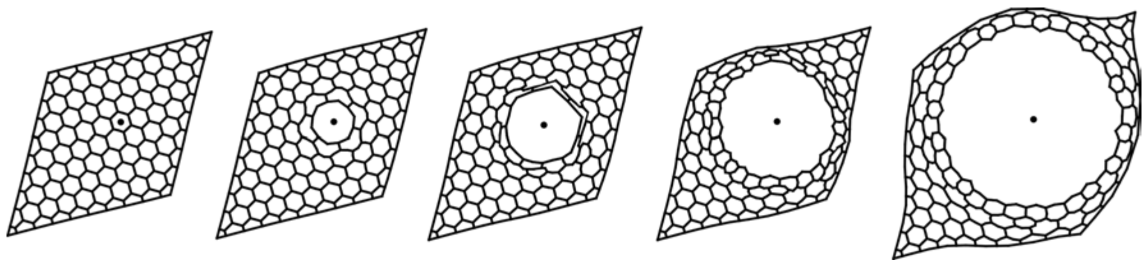


Figure 4.8: A time-series dual view of the simulation of a slowly growing cell (marked with a dot) in an SDS2 form. Spatial and structural cell movements cause the cells to rearrange around the growing cell.

Numerical integration The `SimulatePhysics`(Δt) function updates the state of the cells from the current time, t_0, by an amount Δt to the next time step, $t_1 = t_0 + \Delta t$. The SDS2 prototype solves Equation 4.7 numerically using the midpoint integration scheme (Press et al., 2007, §17.1), which is fast and proved adequate for the experiments that were performed. The scheme approximates the equation

$$f(t_0 + \Delta t) = f(t_0) + \int_{t_0}^{t_0+\Delta t} g(t, f(t))dt$$

as

$$f(t_0 + \Delta t) \approx f(t_0) + \Delta t * g(t_0 + \frac{\Delta t}{2}, f(t_0) + \frac{\Delta t}{2} g(t_0, f(t_0))) \tag{4.9}$$

by assuming the function g is constant over the integration interval, with an approximated value lying halfway between t and $t + \Delta t$. Equation 4.9 can be used to approximate Equations 4.7 and 4.8 which gives an explicit update equation for cell position and velocity.

Other Cell State Changes Cell state changes can also be considered as part of the physical simulator. Like cell position, other cell variables change over time. For example, the cell radius variable $c_r(t)$ is dynamic and is modified with the radius rate of change variable $\frac{dc_r}{dt}(t) = c_{\Delta r}(t)$. An example of the effects of cell growth is shown in Figure 4.8. In SDS2 (and SDS3) it is sufficient to integrate these variables with a simple explicit Euler update, $c_r(t + \Delta t) = c_r(t) + \Delta t * c_{\Delta r}$. A more sophisticated integration scheme may be necessary if more precise cell state dynamics are sought.

4.4 Cell Division

This section compares some strategies for modelling cell division in SDS2 and presents the method used in the SDS2 implementation. Cells that lie on the boundary of an s-morph and cells that lie inside an s-morph are considered separately.

Internal Division An internal cell is completely surrounded by simplexes. Figure 4.9 illustrates two approaches to dividing a cell that has an even number of neighbouring simplexes. Experimentation revealed that the introduction of stabiliser cells (method (d)) works better than the others. In method (c) the configuration is less stable as the shape of the triangles is less regular than in method (d). This physical imbalance means that the daughter cells may move a significant amount immediately after the division. This could lead to the propagation of structural cell movement events, which makes a developmental model difficult to control (see the discussion for more on this §8.5).

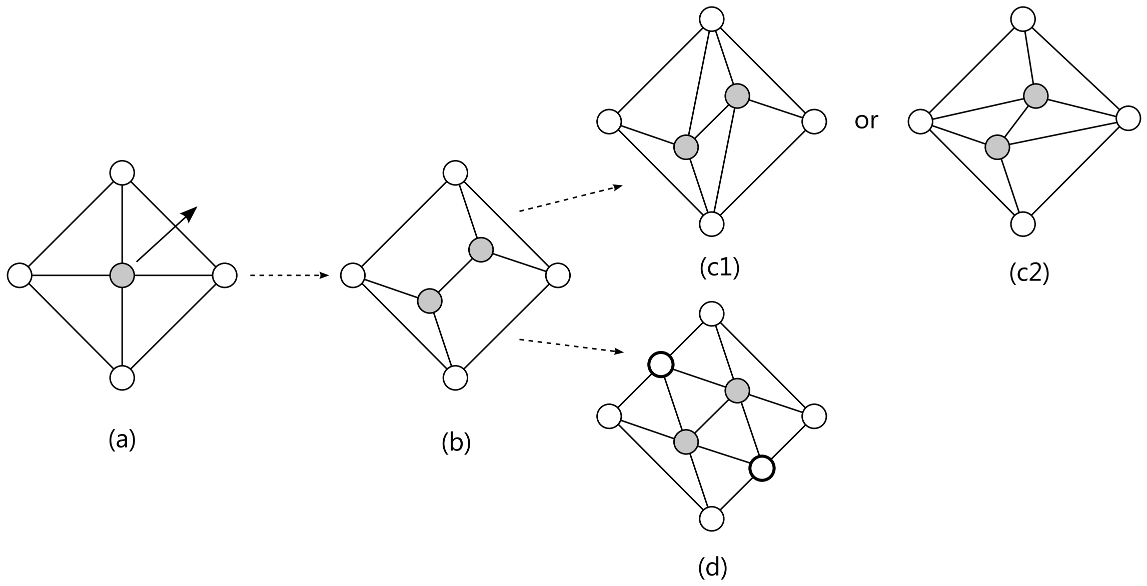
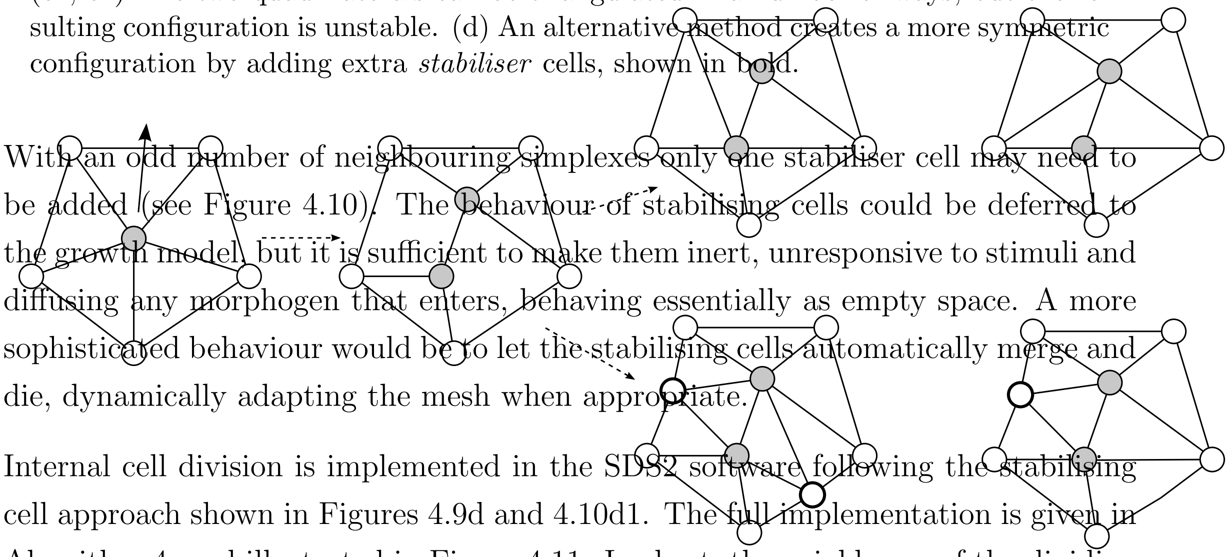


Figure 4.9: Cell division with an even number of neighbours. (a) The shaded cell has elected to divide in the direction shown. (b) The cell is split into two and its neighbours are evenly distributed to the daughter cells resulting in two quadrilaterals. (c1, c2) The two quadrilaterals can be triangulated in a number of ways, but the resulting configuration is unstable. (d) An alternative method creates a more symmetric configuration by adding extra *stabiliser* cells, shown in bold.



With an odd number of neighbouring simplexes only one stabiliser cell may need to be added (see Figure 4.10). The behaviour of stabilising cells could be deferred to the growth model, but it is sufficient to make them inert, unresponsive to stimuli and diffusing any morphogen that enters, behaving essentially as empty space. A more sophisticated behaviour would be to let the stabilising cells automatically merge and die, dynamically adapting the mesh when appropriate.

Internal cell division is implemented in the SDS2 software following the stabilising cell approach shown in Figures 4.9d and 4.10d1. The full implementation is given in Algorithm 4, and illustrated in Figure 4.11. In short, the neighbours of the dividing cell, c, are split into two groups by the half-space defined by c_x and d (Figure 4.11a). The neighbour cells are attached to the daughter cells depending on whether they lie

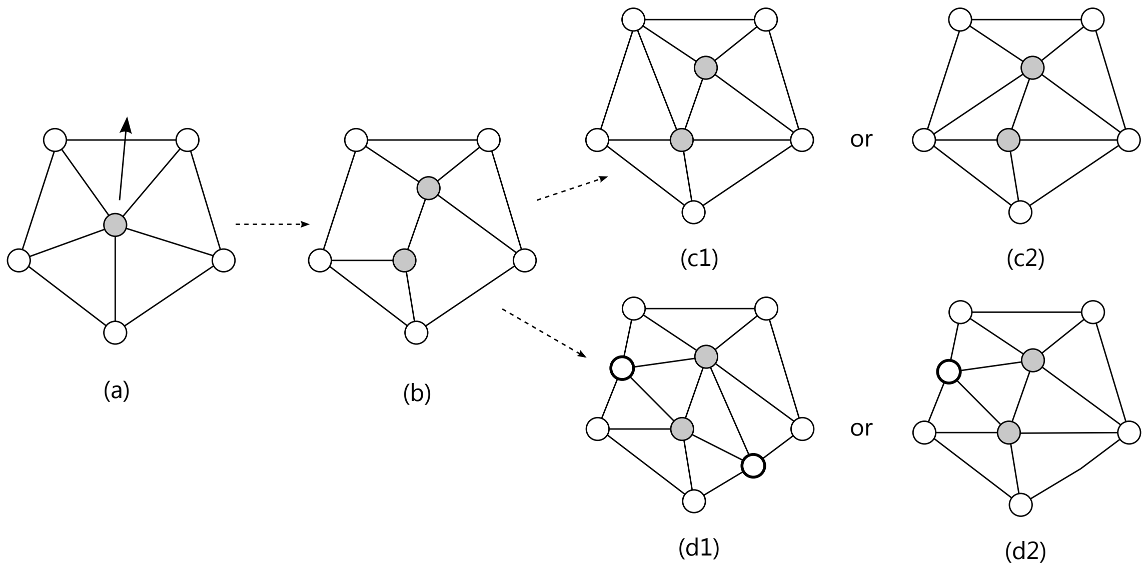


Figure 4.10: Cell division with an odd number of neighbours. (c1,c2) Applying the same approach as in Figure 4.9 results in daughter cells with different numbers of neighbours. (d1) Adding two stabiliser cells does not solve this discrepancy, but (d2) adding just one does. The SDS prototype uses method (d1), however in retrospect (d2) is a more symmetric approach.

in one half-space or the other. As shown in Figure 4.11b, the leftmost and rightmost cells from both half-space groups are found and the edges between l_1 and l_2, and r_1 and r_1, are subdivided with stabilising cells. The final post-division configuration is shown in Figure 4.11c.

Boundary Division A more expressive system is possible if we consider two types of boundary division: *along* the boundary and *away* from the boundary. Division along a boundary allows a surface to grow laterally by adding new cells as shown in Figure 4.12. Division away from the boundary supports the development of extrusions by adding new elements on top of the surface (see Figure 4.13). As with internal division, it may be necessary to add stabilising cells in order to maintain symmetry amongst the daughter cells.

The prototype implementation for division along a boundary follows Figure 4.12 (top row and top path of bottom row), and for division away from a boundary follows 4.13b. The algorithm proceeds as follows: Given a cell c we can identify the two surface neighbours l and r (Figure 4.14a). Depending on the direction of division, d, the algorithm chooses between dividing along or away from the boundary. This is decided by measuring the angle between d and the boundary edges as shown in Figure 4.14b. A threshold angle, θ, determines whether division is performed along (d_1) or away from (d_2) the boundary. Algorithm 5 and Figure 4.14 describe the algorithm in full.

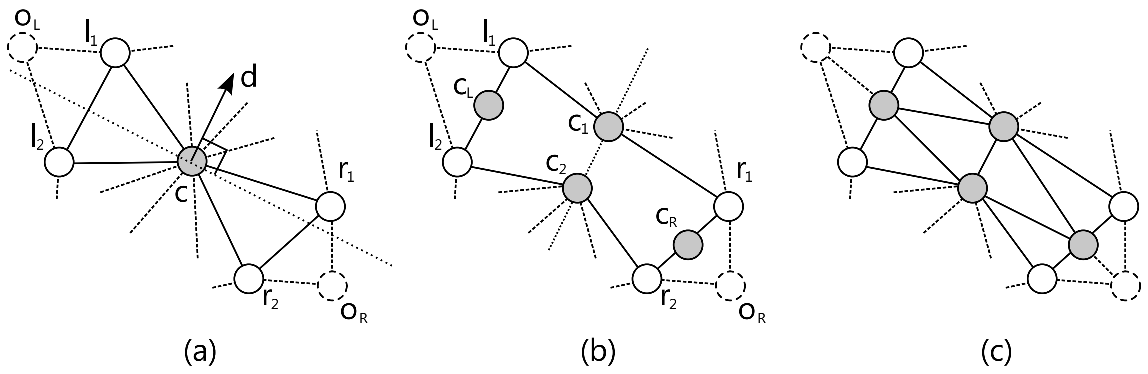


Figure 4.11: Internal cell division in SDS2. (a) Cell c chooses to divide in direction d. The neighbours of c can be split by a line perpendicular to d and intersecting c. Cells o_L and o_R only exist if l_1 and l_2 (and r_1 and r_2) are not both surface cells. (b) Division is performed by first splitting c into two cells c_1 and c_2 that lie along the vector d, and adding the two stabilising cells c_L and c_R as shown. (c) The local configuration after the division operation.

Algorithm 4 `PerformCellDivide(c,d)` when c is internal

```
Let $\epsilon$ be the desired distance between the daughter cells
*Find $l_1, l_2, r_1, r_2, o_L, o_R$ as shown in Figure 4.11a*
$c_1 = c$
Add a new cell $c_2$
$c_{1x} = c_x + d\epsilon, c_{2x} = c_x - d\epsilon$
Let $H = \{x \in \mathbb{R}^2 : x - c_x \cdot d > 0\}$
**for** all $b \in N(c)$ such that $b_x \notin H$ **do**
   Remove edge $(c, b)$
   Add edge $(c_2, b)$
**end for**
*Split edges and add stabiliser cells $c_L$ and $c_R$*
Add new cells $c_L, c_R$
$c_{Lx} = \frac{1}{2}(l_{1x} + l_{2x}), c_{Rx} = \frac{1}{2}(r_{1x} + r_{2x})$
Remove edges $(l_1, l_2)$ and $(r_1, r_2)$
Add edges $(l_1, c_L)$, $(l_2, c_L)$, $(r_1, c_R)$ and $(r_2, c_R)$
*The configuration at this point is shown in Figure 4.11b*
**if** $o_L$ exists **then**
   Add edge $(c_L, o_L)$
**end if**
**if** $o_R$ exists **then**
   Add edge $(c_R, o_R)$
**end if**
Add edges $(c_1, c_L)$, $(c_1, c_R)$, $(c_2, c_L)$, and $(c_2, c_R)$
*The configuration at this point is shown in Figure 4.11c*
```

Algorithm 5 `PerformCellDivide(c,d)` when c lies on the boundary

```
Let $\epsilon$ be the desired distance between the daughter cells
$d_l = \frac{l_x - c_x}{|l_x - c_x|}$
$d_r = \frac{r_x - c_x}{|r_x - c_x|}$
**if** $d \cdot d_r > \cos\phi$ or $d \cdot d_l > \cos\phi$ **then**
    *Perform division along the boundary*
    $c_1 = c$
    Add new cell $c_2$
    $c_{1x} = c_x + d_l \epsilon$
    $c_{2x} = c_x + d_r \epsilon$
    Remove edge $(c_1, r)$
    Add edges $(c_1, c_2)$ and $(c_2, r)$
    *Find L and R as shown in Figure 4.14c2*
    $L = \emptyset, R = \emptyset$
    **for** $b \in N(c)\{l, r\}$ **do**
        **if** $\frac{b_x - c_x}{|b_x - c_x|} \cdot d_l > \frac{b_x - c_x}{|b_x - c_x|} \cdot d_r$ **then**
            Add $b$ to $L$
        **else**
            Add $b$ to $R$
        **end if**
    **end for**
    **for** $b$ in $R$ **do**
        Remove edge $(b, c_1)$
        Add edge $(b, c_2)$
    **end for**
    *Find $c_{rl}, c_{lr}$ as shown in Figure 4.14c3*
    *Compute $L_\theta$ and $R_\theta$ as shown in Figure 4.14c3*
    **if** $L_\theta < R_\theta$ and $c_{lr}$ exists **then**
        Add edge $(c_1, c_{lr})$
    **else if** $c_{rl}$ exists **then**
        Add edge $(c_2, c_{rl})$
    **end if**
**else**
    *Perform division away from the boundary*
    *Re-orient d to point away from the s-morph as shown in Figure 4.14d2*
    $c_1 = c$
    Add new cell $c_2$
    $c_{1x} = c_x - d'\epsilon, c_{2x} = c_x + d'\epsilon$
    *Add edges $(c_2, l)$, $(c_2, r)$ and $(c_1, c_2)$ as shown in Figure 4.14d3*
**end if**
```

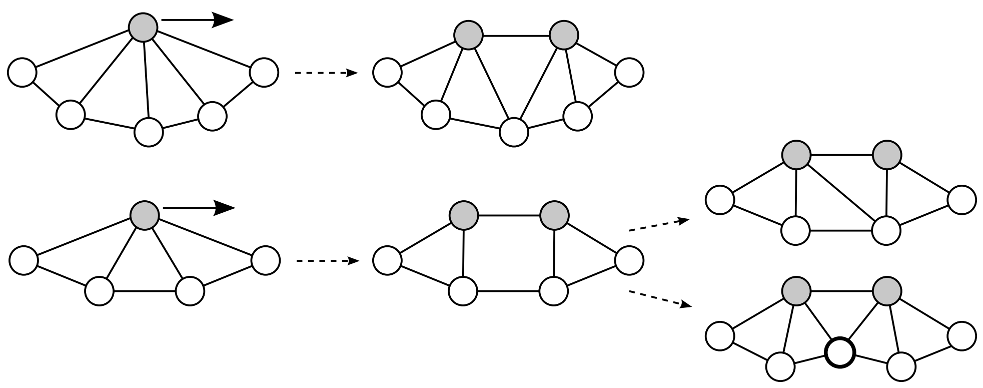


Figure 4.12: Cell division along a boundary. The upper figure demonstrates the trivial division of a cell with an odd number of neighbours. The lower figure illustrates a non–triangular region that can form when dividing, which can be re-triangulated either asymmetrically or with a peripheral stabilising cell.

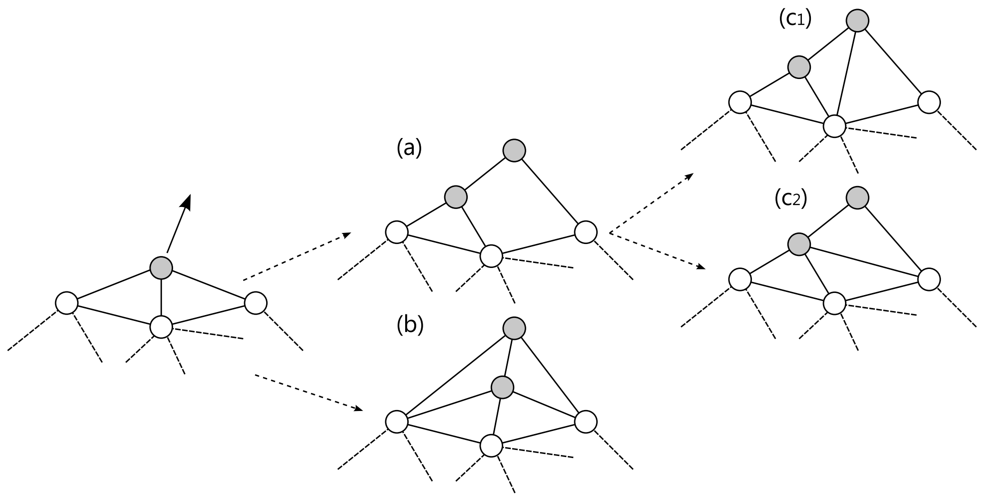


Figure 4.13: Cell division away from a boundary as (a) one-sided or (b) symmetric operations. A one-sided division results in a quadrilateral which must then be triangulated into either (c1) or (c2). Method (b) is used in the SDS2 prototype.

Figure 4.15 shows an SDS2 simulation in which consecutive cell divisions occur. Each frame is shown in mesh view and cell dual view. From the top left: (a1,b1) The highlighted cell has elected to divide horizontally. (a2, b2) The instant it divides two daughter cells and two stabiliser cells are created. The stabiliser cells are shown in dark gray in b2. (a3,b3) The highlighted cell elects to divide diagonally. (c1,d1) The four new cells and the resulting mesh are shown. (c2-c3,d2-d3) Another cell chooses to divide. (e1-e3,f1-f3) After the last division the s-morph is unbalanced, the light gray cell in (f2) is pushed towards the dark gray cell. In (e2) we see the cell is just about to cross over an edge. (e3,f3) The result of performing a cell move (both grey cells are now adjacent).

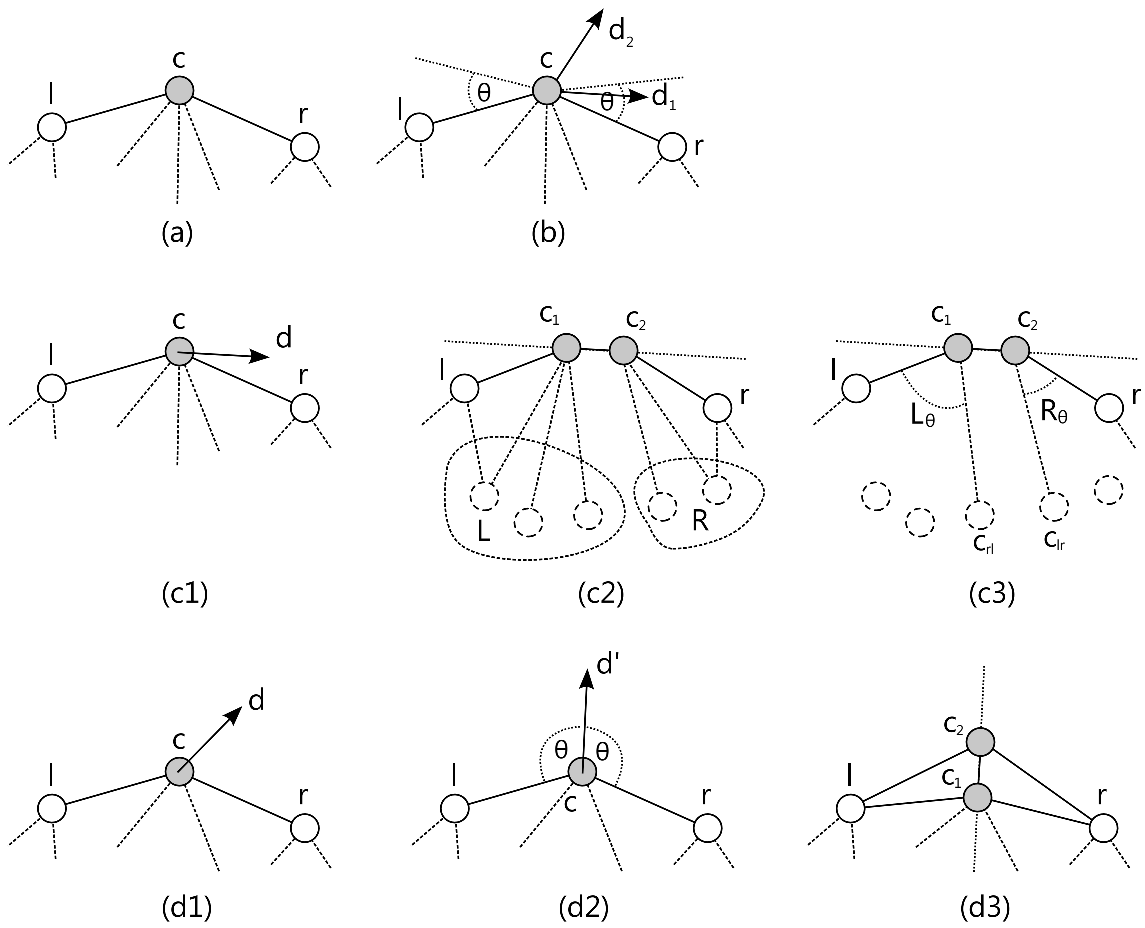


Figure 4.14: Cell division on an s-morph boundary (see Algorithm 5). (a) A cell c and its left and right surface neighbours l and r. (b) Division along or away from the boundary is determined by whether the direction of division lies within a threshold angle, for example direction d_1 will result in division along the boundary and direction d_2 . (c1) A cell c divides along the boundary in direction d. (c2) The cell is split into two daughter cells c_1 and c_2 as shown. The sub-surface neighbours of c are split into two groups L and R which are reattached to the daughter cells. (c3) Finally, L and R must be joined, this is done by identifying cells c_{rl} and c_{lr} as described in the algorithm. (d1) A cell, c, may also divide away from the boundary. (d2) The division direction is normalised to bisect the edges cl and cr. (d3) Cell c is split into two daughter cells c_1 and c_2 and structured as shown.

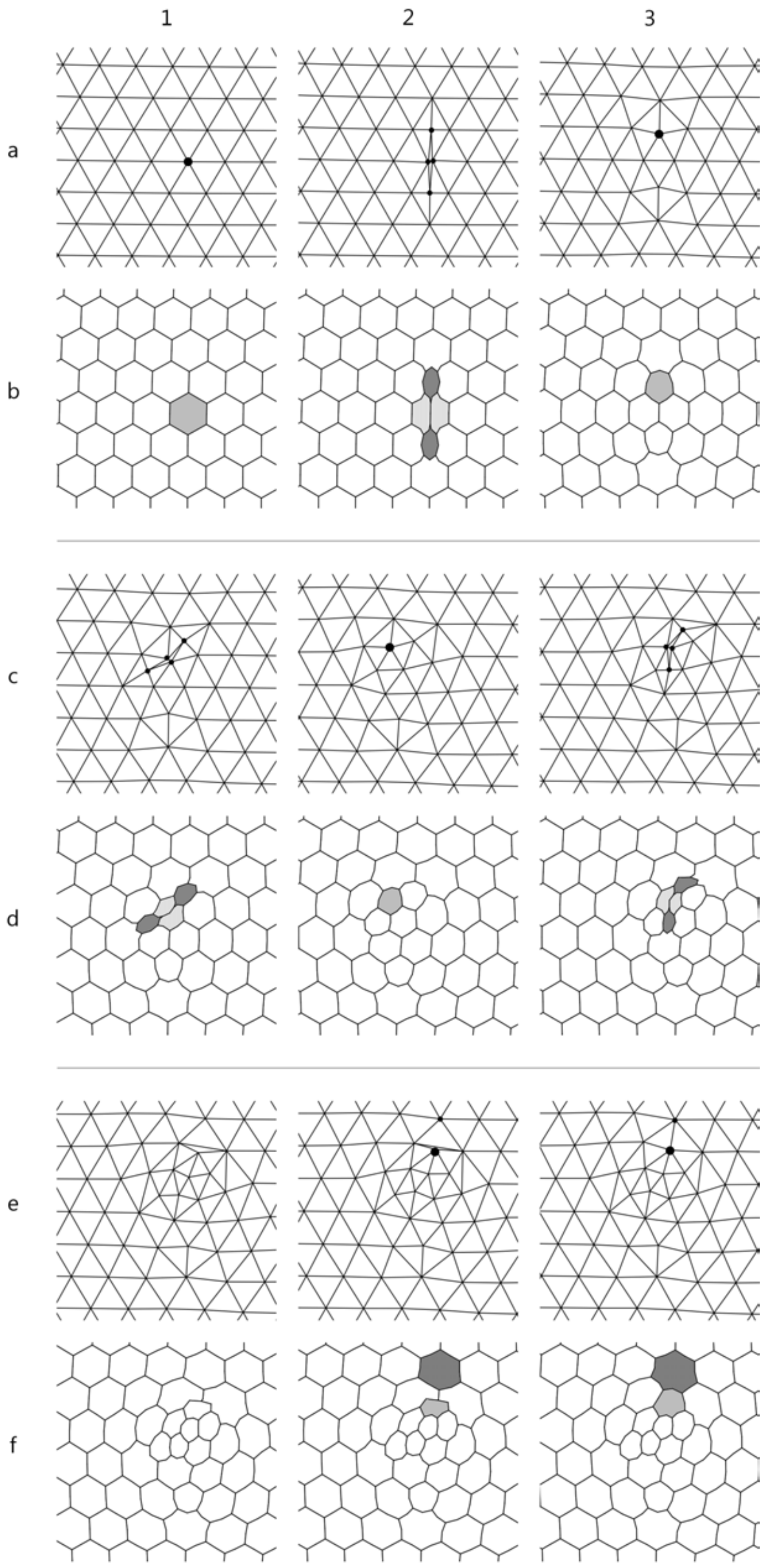


Figure 4.15: (left to right, top to bottom) A time series from an SDS2 simulation in which three cell divisions and one cell movement operation occur. Each frame is shown in mesh view and cell dual view. See the text for the description.

4.4.1 Cell Division as Simplex Subdivision

The implementation for SDS2 cell division works well for triangular meshes, however it does not generalise easily to tetrahedral meshes. Consider the internal cell division algorithm (Algorithm 4). A key step in this algorithm involves dividing the set of neighbours of a cell into two halves, attaching one daughter to one half and the other daughter to the other half, removing the triangles which lie in the boundary between the halves, and then restructuring the space between the two halves. The obvious analogue in three dimensions is to split the set of neighbouring tetrahedra into two halves, connect the daughter cells to their respective halves, remove the tetrahedra which lie in the boundary between the halves, and then restructure the space between the two halves. A brief examination of this problem reveals that there are many different cases that can arise when splitting a set of tetrahedra into two halves — the 2D case is trivial in comparison. Appendix B considers this problem in more detail.

Therefore, with generality in mind, a division algorithm that could work in 2D *and* 3D was sought. This led to the idea of using simplex subdivision to model cell division: *when a cell divides, subdivide a neighbouring simplex in the direction of division.* Simplex subdivision applies to all simplexes: tetrahedra, triangles and edges. Algorithm 6 outlines the method for subdividing a neighbouring face, which also applies to SDS3 (see Algorithm 9 (p123)). Figure 4.16 demonstrates some different cases of simplex subdivision in SDS2. Edge subdivision adds an extra degree of accuracy when a cell wishes to divide in a particular direction.

Algorithm 6 Simplex subdivision

```
  Input: $c \in C$, $d \in \mathbb{R}^2$
  Choose the $f \in F$ that lies in direction $d$, where $c \in f$
  Subdivide $f$ (see Figure 4.16a) by adding a new cell $c_2$ at the center.
  Replace $c$ with $c_1$
  Output: $c_1$ and $c_2$
```

Of the cell division guidelines proposed in §3.5.2, simplex subdivision follows rule 1 (daughter cells are neighbours) and loosely follows rule 2 (directional division). However, it is obvious that simplex subdivision does not result in topologically balanced structures (rule 3). For example, in Figure 4.16a, we have $|N(a)| = 7$, while $|N(b)| = 3$. In general, with triangular subdivision, we always have $|N(a)| = |N(c)| - 1$ and $|N(b)| = 3$ and with edge subdivision $|N(a)| = |N(c)|$ and $|N(b)| = 4$. This asymmetry may result in topologies that are difficult to work with (see §6.4).

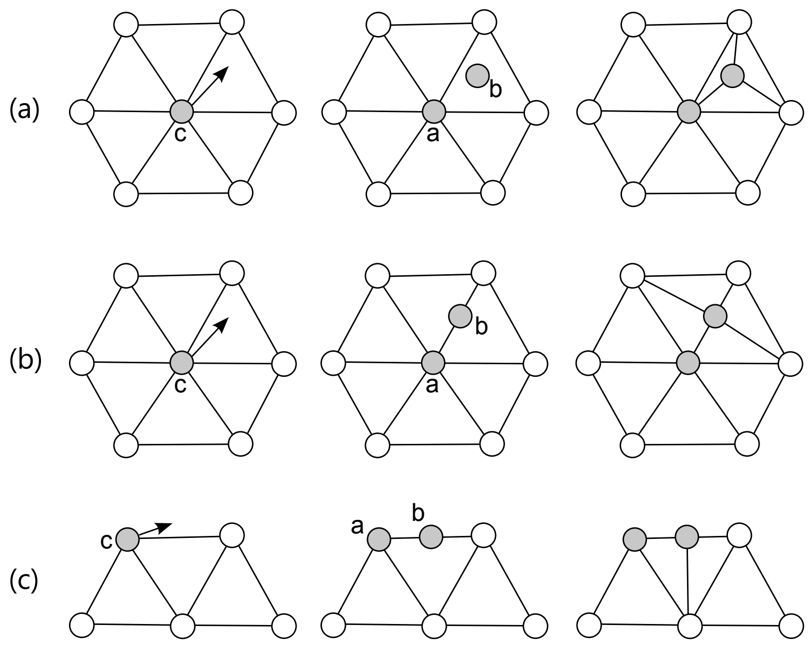


Figure 4.16: Cell division modelled as simplex subdivision. Each of the three rows demonstrates a different case, in which a cell c divides in the direction shown and is replaced by two daughter cells a and b. (a) To perform triangle subdivision, choose the triangle, t, adjacent to c that lies in the direction of division. Rename c to a and add a new cell, b, in the center of t. Replace t with three triangles as shown. (b) To perform edge subdivision, choose the edge, e, that lies closest along the direction of division. Let $a = c$, as above, and split e with the new daughter cell, b, as shown. (c) Edge subdivision can also be performed on the boundary as shown.

4.5 Cell Movement

The cell movement transformation in SDS allows the structure of an s-morph to adapt to the movement of cells (§3.5.3). As cells move around within an s-morph they may occasionally come into contact with, or cross over, the edges of the mesh. When this occurs the mesh may become invalid (due to overlapping triangles or edges), a situation that must be avoided. An example of a series of SDS2 cell movements are shown in Figure 4.17.

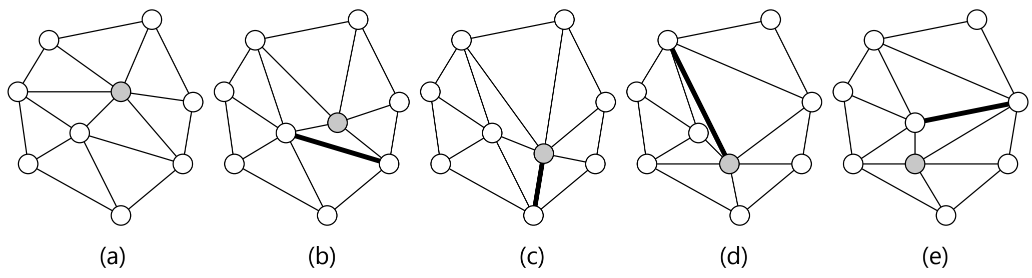


Figure 4.17: An example of a series of SDS2 cell movements. (a) A cell (shaded) moves within an s-morph, causing a series of structural adaptations. (b) The cell is just about to intersect with an edge (bold). (c) At the instant it collides with the edge, the edge is flipped to the new configuration shown. (d) The cell continues to move, causing a collision between an edge and a different cell, resulting in (e).

The cell movement transformation in SDS2 can be performed simply as follows: When a cell crosses over an *internal* edge, an edge-flip is performed (Figure 4.18). The edge-flip operation was first proposed as a model for cell movement by Matela and Ransom (§4.1.3). They considered cell movement as the addition or removal of edges, which in a triangulated graph can be reduced to the *exchange* of edges. Duvdevani-Bar and Segel (1988) extended this model by considering the positions of cells and allowing cells to move through space as a result of undergoing neighbourhood exchanges. The primary difference between their model and SDS2 is that in SDS2 the operation is performed only when a cell crosses over an internal edge. This crossing is detected by performing a line intersection test between the edge and the line segment between the points $c_x(t)$ and $c_x(t + \Delta t)$.

If a cell crosses a boundary edge, then the edge cannot be flipped. In this case, SDS2 simply splits the crossed edge with the cell, as is illustrated in Figure 4.19. The special case where both the moving cell and the crossed edge lie on the surface (Figure 4.19b) should be handled appropriately. This case never came up in the experiments performed; however, it could be dealt with by using the collision detection and handling in the physical simulator.

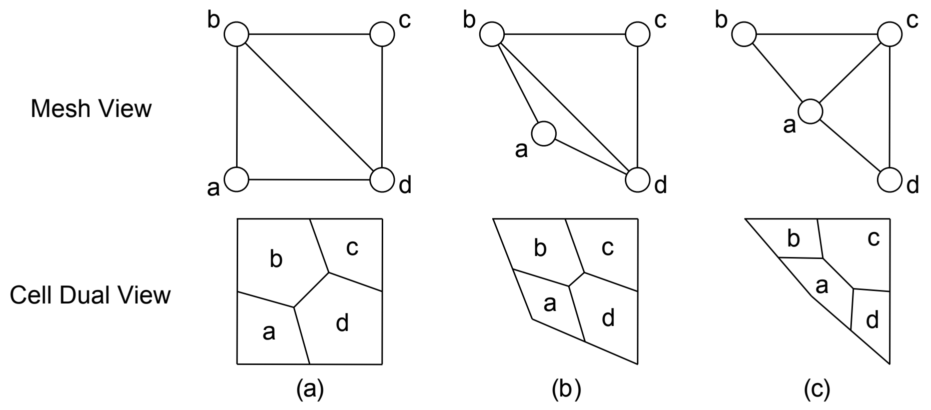


Figure 4.18: A cell movement transformation in SDS2. (a) The mesh of an s-morph with four cells a, b, c, and d and the cell dual view of the s-morph. (b) Cell a at the instant before it crosses over the edge bd. (c) The movement transformation *flips* the edge bd replacing it with a new edge ac. The cell move is best visualised using the cell dual view, cell a comes into contact with cell c, while cells b and d are separated.

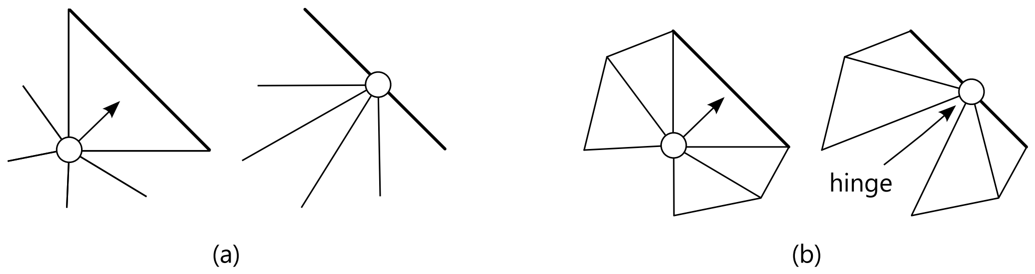


Figure 4.19: SDS2 boundary cell movement. (a, left) An internal cell moves towards a boundary edge. (a, right) The cell movement transformation bisects the edge with the cell. (b) If a boundary cell is pushed into a boundary edge, naïvely splitting the edge will result in a hinge and therefore an invalid s-morph structure.

In practice it was found that the time step used in the experiments was small enough such that multiple crossings did not occur simultaneously. For simplicity, any detected movement is assumed to occur at time $t + \Delta t$ and hence there is no need to rewind time before performing the movement operation. `TimeOfFirstMovement()` thus always equals Δt and hence `RewindSimulation`$(\widetilde{\Delta t})$ is unnecessary. If multiple crossings occur, or if greater accuracy and robustness is required, then these components would need to be implemented. The SDS3 prototype implements these methods (§6.5), and the techniques used could be easily applied to an SDS2 implementation. A cell movement occurring in an SDS2 simulation is shown in Figure 4.15(e2-3).

4.6 Morphogen Model

In SDS, morphogens are contained within the cells, and flow between adjacent cells (refer to Figure 3.6b). In SDS2 this is implemented in the framework through the `SimulateMorphogens`(Δt) function. The growth program specifies a set of m morphogens $\Phi = \{\phi_1, \ldots, \phi_m\}$, each with corresponding diffusion and decay coefficients, D_{ϕ_i} and C_{ϕ_i}. Each cell contains a continuous amount of each morphogen. The morphogen equation is spatially discretised over the triangular mesh following the assumptions given in §3.4.2. Recall the morphogen equation is:

$$\frac{\partial c_\phi}{\partial t} = D_\phi \nabla^2 c_\phi - C_\phi c_\phi \tag{4.10}$$

To compute $\nabla^2 c_\phi$ at each cell for some timestep, the following general discretisation is used:

$$\nabla^2 c_\phi = \sum_{n \in N(c)} \gamma(n, c)(n_\phi - c_\phi), \tag{4.11}$$

where $\gamma(n, c)$ is a weighting function. $\gamma(n, c) = (|N(c)||n_x - c_x|^2)^{-1}$ is used in the simulations shown in the next chapter, but different $\gamma(n, c)$ functions will give different morphogen dynamics. In retrospect, $\gamma(n, c) = |c_x - n_x|^{-1}$, as used in SDS3 (see Equation 6.27), is a much simpler function and still allows the formation of morphogen gradients. The `SimulateMorphogens`(Δt) function updates the morphogen values using an explicit Euler step:

$$c_\phi(t + \Delta t) = c_\phi + \Delta t(D\phi \nabla^2 c_\phi - C_\phi c_\phi), \tag{4.12}$$

where c_ϕ is the value at time t, and the result is clamped to the range $[0, \mathrm{vol}(c)]$.

The morphogen simulation performed is somewhat naïve but produces acceptable results (e.g., Figure 5.13). In particular, the Euler integration scheme does not conserve the morphogens based on the continuous equation. In practice, however, the morphogens are crudely used to either set up gradients and give cells an axis of division, or to demarcate spatial zones using the gradients. In this case, the accuracy of the simulation is not an issue. If higher accuracy is required then the resolution of the triangular mesh can be increased or a more conservative integration scheme could be used (e.g., fourth-order Runge-Kutta (Press et al., 2007, §17.1)).

4.7 Summary

This chapter presented the details of a 2D implementation of SDS, called SDS2. In 2D, an s-morph is modelled as a system of circular cells connected together with a triangular mesh. Algorithms were given for performing cell division and cell movement on a triangular mesh, and it was shown how a mass-spring model can be coupled to the cell system. Finally, it was described how the morphogen equation can be discretised on a triangular mesh. All these components form the machinery of the 2D developmental system. The next chapter shows how to use this system in order to generate organic structures.

Chapter 5

Experiments in 2D

The previous chapter formalised the machinery of SDS2. This chapter will demonstrate how to use SDS2 to generate developing limb forms and stripe patterns. Early on in the research the SDS2 prototype was used to explore some basic ideas in form generation. This exploration fed back into the design of SDS and demonstrated that SDS could generate the desired forms, leading to the eventual development of SDS3.

The first experiment presented in this chapter is the limb bud model (§5.1), which demonstrates that a basic cell behaviour, acting in parallel with the physical and morphogen models, can generate some interesting forms and cellular configurations. The second experiment, the formation of striping (§5.2), illustrates how a simple model of morphogen flow can be used to coordinate the spatial patterning of cells. These experiments demonstrate how SDS uses morphogen patterns, local information and geometric operations to construct forms.

An important element of SDS and any developmental system is the relationship between cell-level (or local) processes and global structures. This chapter demonstrates how, by carefully designing and controlling coordinated cell-level processes, macro-level structures and morphogen distributions can be generated, thus addressing the problem of emergence, one of the key aims of this research (§1.2). Moreover, it is shown how locally acting processes can be grouped together as "modules" that can then be repeated within a single design, or re-used in other, new designs. For example, the limb module can be re-used, simply by "implanting" the limb at various locations within a mesh. Furthermore, there are a number of advantages gained through the emergence of the limb, and other modules. Firstly, the growth process typically creates a natural structural coupling between modules, i.e., there isn't a clear structural delineation between limb and body. Secondly, variations on the module can be obtained by varying the parameters of the growth model, allowing, for example, the growth of wider limbs. Lastly, each of the processes of the limb model are sensitive to local structure and environmental conditions. If numerous limb tips are implanted within a mesh the environmental-sensitivity of the processes results in limbs that are similar, but not exactly the same (demonstrated by the variation between starfish arms in Figure 4.1). These advantages are all benefits of the process-based, embedded approach of SDS.

The two growth models are now presented in detail.

5.1 A Study of Limb Bud Formation

This section discusses how a biological model of limb bud development in chick embryos was implemented in SDS2 and how it leads to the emergence of a rich set of organic geometries. Aspects of the model, including re-use, module boundary, repetition, and variation are also considered.

5.1.1 Limb Growth in Chicks

Limb growth in early chick embryogenesis is the result of a coupled reaction between the *epithelium* (a proto skin) and *mesenchyme* (free floating cells beneath the skin) (Gilbert, 2006, Chapter 16). The development involves interactions between a region on the epithelium called the *apical ectodermal ridge* (AER) and the underlying mesenchyme. Broadly, the AER diffuses a protein, *Fgf8*, which induces the underlying mesenchyme to proliferate towards it. The proliferating mesenchyme diffuses a protein, *Fgf10*, which induces the AER to produce more Fgf8. This feedback loop causes a cluster of cells to proliferate in one direction, resulting in the growth of a primitive limb (see Figure 5.1.)

The key elements of this model are:

- The directed proliferation of a cluster of cells towards a target cell,
- The stimulation of the proliferating cluster using morphogen gradients and activation thresholds, and
- A feedback loop which drives the process.

The following results demonstrate that these elements are sufficient to create simple limb-like growths in SDS2.

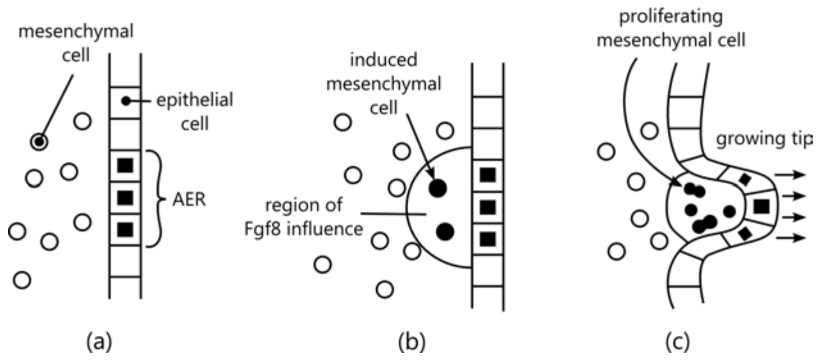


Figure 5.1: A model of limb bud development in chicks. (a) The AER cells (the black squares) are initialised in the epithelium (primitive skin cells). (b) The AER releases a protein, *Fgf8*, which induces the nearby mesenchyme (free floating internal cells) to (c) grow and divide towards the AER, resulting in a feedback cycle and the proximal-distal (outwards) growth of a limb.

5.1.2 Growth Model

The limb model was implemented in SDS2 by modelling the two proteins with morphogens and capturing the cell behaviour with a set of simple rules. The key features of the growth model are:

- There are two morphogens ϕ_8 and ϕ_{10} which model the two proteins Fgf8 and Fgf10,
- Cells can create morphogens at a linear rate: $\frac{dc_\phi}{dt} = c_{\Delta\phi}$,
- A cell can access the local gradient of each morphogen, $\nabla\phi_i$, and
- An additional cell variable, $c_t \in \{\texttt{AER}, \texttt{E}, \texttt{M}\}$, is used to distinguish between the AER, epithelium and mesenchymal cells.

The cell type variable is necessary to distinguish the behaviour of the AER from the surrounding cells. The distinction between E and M provides control over the proliferation rates of the boundary and internal cells. The AER cells are specified in the initial conditions, and for all non AER cells $c_t = E$ if $c_{surface} = true$ and $c_t = M$ otherwise. The cell behaviour is modelled with the set of rules in Table 5.1.

Depending on its type a cell may execute the AER, E, or M rules. The rule-sets are evaluated at each call of `RunCellPrograms`(Δt) and if a condition is satisfied then the action is performed. For example, Rule $\texttt{AER}_1$ dictates that a cell of type `AER` with a ϕ_{10} concentration greater than $\texttt{AER}_{thres}$ produces the morphogen ϕ_8 at a linear rate of $\triangle\phi_8$. In total, there are nine[1] model parameters: $\texttt{AER}_{thres}$, $\texttt{M}_{thres}$, $\texttt{E}_{thres}$, $\triangle\phi_8$, $\triangle\phi_{10}$, $\triangle r_M$, $\triangle r_E$, $\texttt{M}_R$, and $\texttt{E}_R$. The effect of the parameters in this system are dependent on subtleties of the implementation, and hence the parameter

[1]This is a lot of parameters! In the SDS3 experiments this model is refined considerably (§7.1).

values used in the experiments presented here are unimportant. Through trial and error, it was found to be relatively easy to locate a viable range of parameters for growing a limb. Once a set of parameters is found it is interesting to consider the effect of varying the parameters, as discussed later (§5.1.3.3).

Table 5.1: Cell behaviour rules for the limb growth model.

rule	condition	action
$\texttt{AER}_1$	$c_t = \texttt{AER}$ & $c_{\phi_{10}} > \texttt{AER}_{thres}$	$c_{\Delta\phi_8} = \triangle\phi_8$
$\texttt{M}_1$	$c_t = \texttt{M}$ & $c_{\phi_8} > \texttt{M}_{thres}$	$c_{\Delta\phi_{10}} = \triangle\phi_{10}$, $c_{\Delta r} = \triangle r_M$
$\texttt{M}_2$	$c_t = \texttt{M}$ & $c_r > \texttt{M}_R$	divide($\nabla\phi_8$)
$\texttt{E}_1$	$c_t = \texttt{E}$ & $c_{\phi_8} > \texttt{E}_{thres}$	$c_{\Delta r} = \triangle r_E$
$\texttt{E}_2$	$c_t = \texttt{E}$ & $c_r > \texttt{E}_R$	divide($\nabla\phi_8$)

5.1.3 Results and Discussion

After much experimentation the formation of limb-like structures was achieved (Figure 5.3). This form generation capability was explored in isolation, within the context of re-use, and the effects of the parameters were considered. This section presents and discusses the results of this exploration and demonstrates the use of the limb bud model in the generation of organic starfish-like forms.

5.1.3.1 Limb Growth in Isolation

The first experiment explored the limb bud model in an isolated context (much like an experimental control). The experiment began with a rectangular configuration of cells (Figure 5.2), a single cell of which was manually initialised with a full concentration of ϕ_{10} and designated as an `AER` cell. The simulation sequence shown in Figure 5.3 demonstrates that this model results in simple limb-like forms (also see animation SDS2/3). The limb bud model reveals a general method for directing growth: local induction with feedback and directed proliferation. The two way induction allows a cluster of cells to coordinate their actions and the form that develops is a result of this cluster forcing the epithelium outwards. The organic configuration of the cells is a direct consequence of the physical simulation and the dynamic reconfiguration.

During growth, the limb module closely couples itself with the body it grew from resulting in an organic interface. Figure 5.4 illustrates that the coupling between the geometry of the body and limb is complex. These results show that SDS is capable of generating organic module boundaries, and thus can potentially support the generation of complex modularity see in the forms that inspired this research

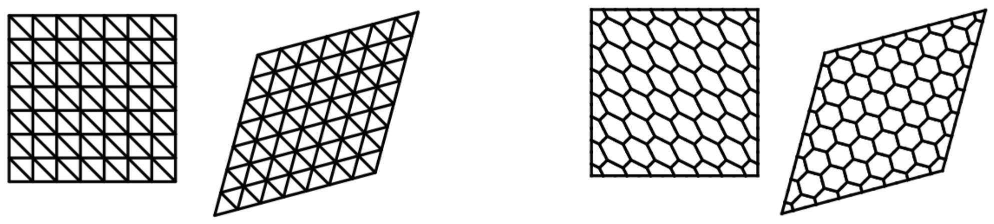


Figure 5.2: The result of the initial perturbation of the starting structure of the s-morph. The mesh is shown in the two figures on the left and the cell dual in the two on the right. The initial s-morph (the square) is not physically stable and so transforms into the lower energy configuration (the diamond). In the lower energy configuration the triangles in the mesh become equilateral, or equivalently, the cells in the dual view become regular hexagons.

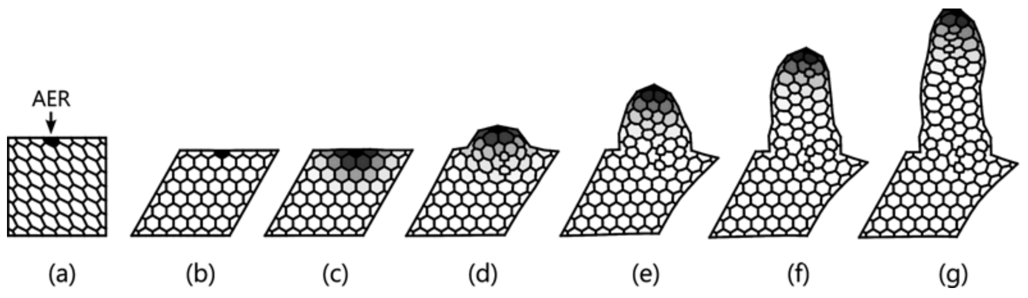


Figure 5.3: Simulation of limb growth. (a) The initial form is rectangular with an AER cell at the top. The form is quickly forced into (b) a minimal energy configuration by the physical model. (c) ϕ_8 (shown as shading) begins to diffuse into the surrounding region. The nearby mesenchyme and epithelium begins to proliferate towards the AER, causing (d) a bump to appear. (e-g) During the feedback loop the growing tip is pushed away from the proliferating cluster of cells underneath it.

(§1.2). In the implemented model there is no distinction between the adhesion within the mesenchyme and epithelium; this may be the cause of the tumour-like appearance of the growths. In reality the epithelium is tightly formed whereas the mesenchyme contains free floating cells. This could be incorporated into the model by assigning spring strengths according to region, but was not explored in this research.

5.1.3.2 Growing Multiple Limbs

In biological development, a single gene can be activated in different locations within an organism. This provides a mechanism for modular re-use and repetition. In a creative system, the design of form via the specification and re-used of modules can be extremely useful, and SDS supports this. The experiments presented here demonstrate how a limb can be re-used in a design to produce multiple limbed forms. The simulations are initialised with an s-morph which includes multiple AER cells located at different positions.

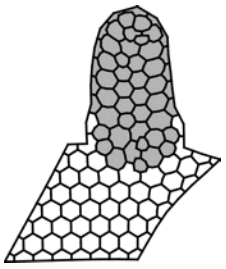


Figure 5.4: A snapshot in time of simulated limb bud growth. All cells that were involved in proliferation towards the AER are shaded, revealing the complex organic boundary of the module.

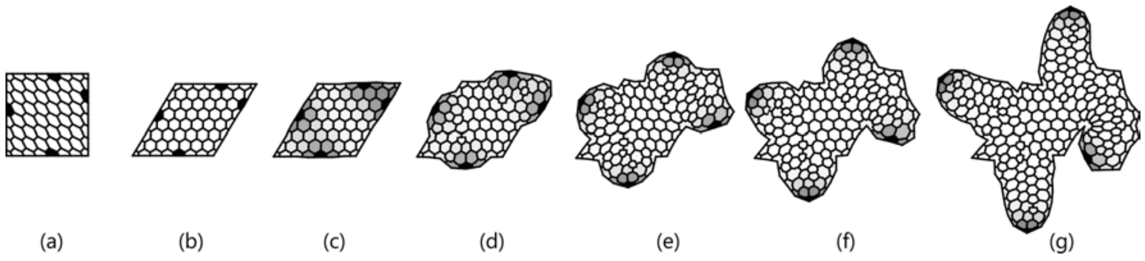


Figure 5.5: The simultaneous growth of four separate limbs. (a) The form is initialized with four AER cells. (b-f) Each separate AER region induces a nearby cluster of cells to proliferate and set up the feedback loop. (g) The last form contains four similar limbs. The rightmost limb is distinct in its curved character. It may have formed like this due to the location of its AER being initially very close to both the corner of the geometry and the topmost AER.

Figure 5.5 shows an initial experiment in which four limbs are grown simultaneously from a diamond-shaped body. In this experiment the cells which aren't involved in proliferation remain inactive. A similar experiment, shown in Figure 5.6, begins with a different geometry, and this time non-proliferating cells were instructed to grow, resulting in tapered limbs (also see animation SDS2/5).

These experiments demonstrate that SDS2 supports modularity, organic symmetry, and physicality. The arms of the starfish (Figure 5.6) illustrate that modules can be designed and then applied in different locations. The different initial contexts of the arms and complex interactions during growth, give rise to subtle variation amongst the modules and results in an imperfect, organically symmetric form. The starfish also has a solid appearance caused by the bending of the limbs and the low energy arrangements of the cells (also see Figures A.2 and A.3). The limb bud model can be implanted in arbitrary geometry, not just radially symmetric forms. Figure 5.7 illustrates one such experiment, in which limbs were grown from a thin stem.

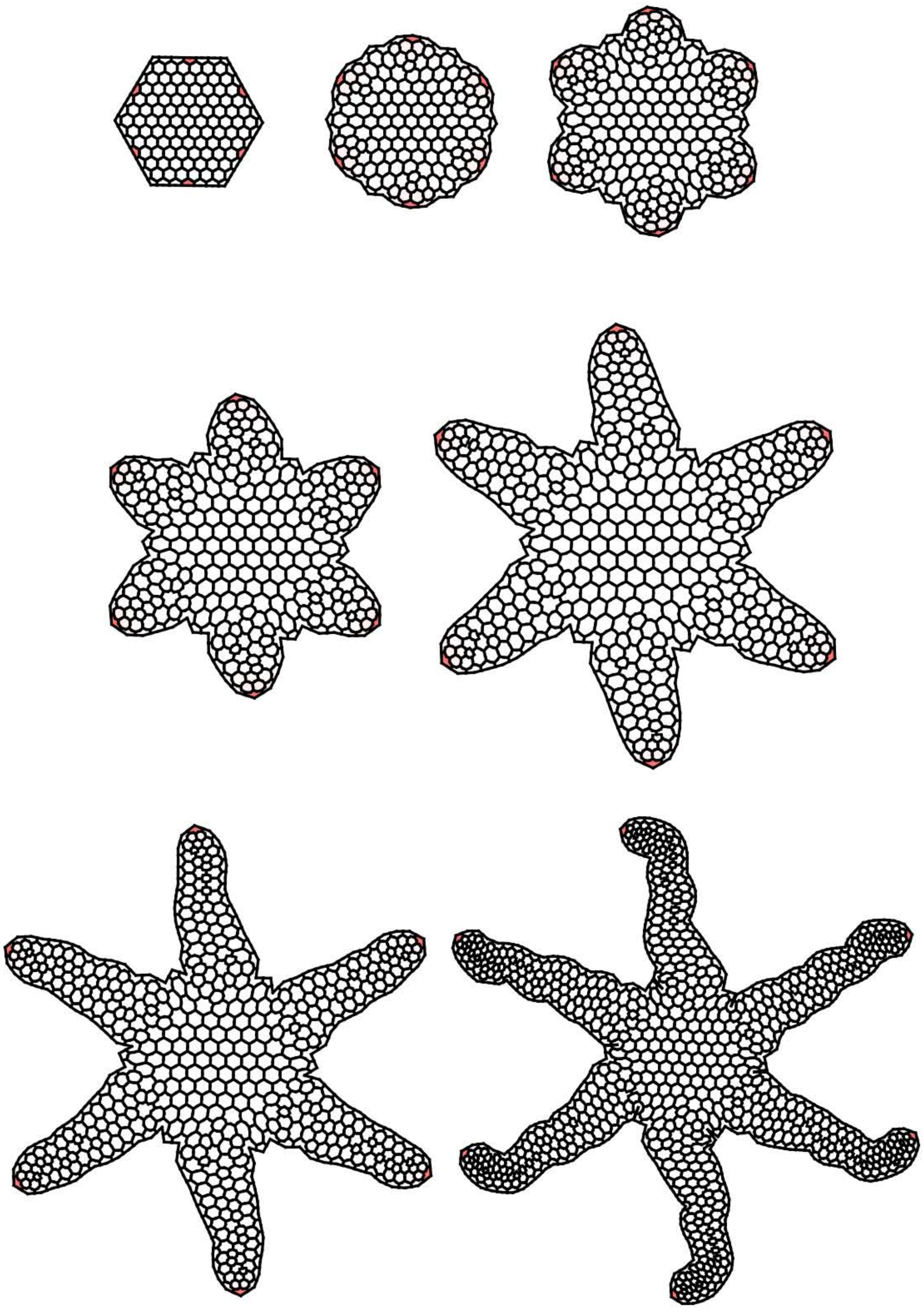


Figure 5.6: (left to right, top to bottom) The developmental sequence of a starfish-like form grown from a hexagonal s-morph with six implanted limb tips. The non-proliferating cells are instructed to grow, which results in a tapering of the limbs.

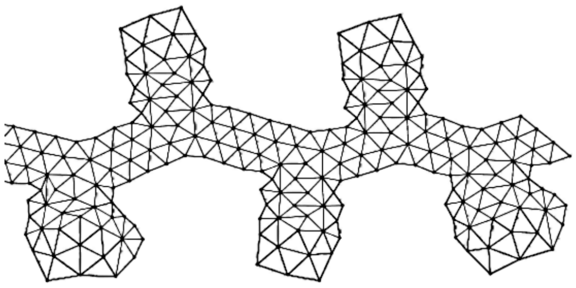


Figure 5.7: An experiment in which limb tips are implanted onto a thin stem, resulting in a primitive plant-like form.

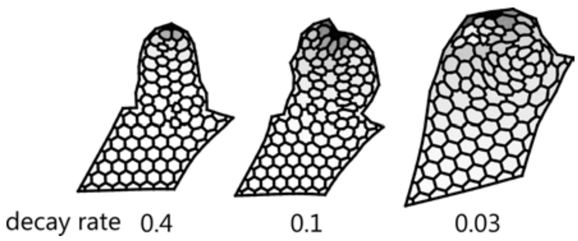


Figure 5.8: The effect of different morphogen decay rates on limb growth. Three separate simulations were conducted with different ϕ_8 decay rates. As the decay rate is decreased the range of ϕ_8 (and hence the proliferation zone) is expanded, resulting in the formation of larger limbs.

5.1.3.3 Exploring the Parameters

Adjusting the parameters of the model can lead to different characteristics in the developing form. For example, the growing tip cluster can be increased in size by decreasing the rate at which ϕ_8 decays. Experiments indicate that this leads to larger growths which are the result of a larger proliferating zone in the mesenchyme (see Figure 5.8).

By varying other parameters, some interesting relatives of the starfish can be generated (see Figure 5.9). These forms were located by performing a sweep of all the limb-bud parameters within certain ranges, using the off-line simulator described in §8.6.2, and then selecting a representative sample. These results demonstrate that qualitative aspects of the starfish model, such as limb width and curvature, can be influenced by varying the parameters of the growth model. In Figures 5.9 (c), (d), and (e), we can see that some sets of parameters lead to limb growth failure either in all limbs bar one, in all limbs, or in two limbs. Figure 5.10 illustrates an earlier experiment in which the rate of growth within the proliferation region was too high, resulting in *tumourous* growths. These experiments show how large-scale structures can emerge through coordination of local geometric modifications. The next section explores how cells can infer spatial information from a morphogen gradient.

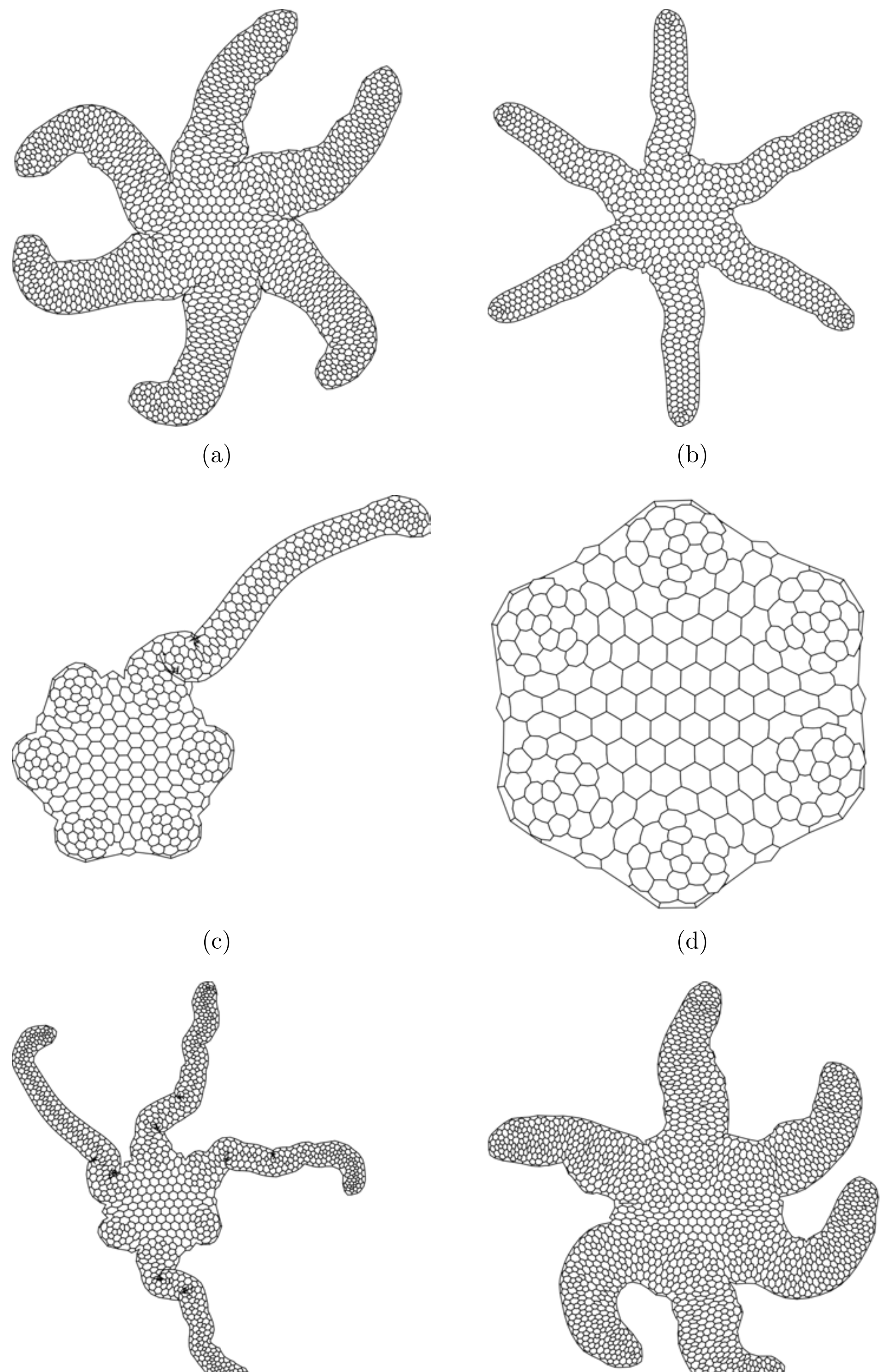


Figure 5.9: Some relatives of the starfish grown using different sets of parameters. (a) Larger limbs can be created by increasing the region of proliferation. (b) Less tapering by decreasing the base rate of growth. (c,d,e) Some sets of parameters can lead to only one limb growing, no limbs growing, or four limbs growing. (f) A relative of (a) has the same large limbs.

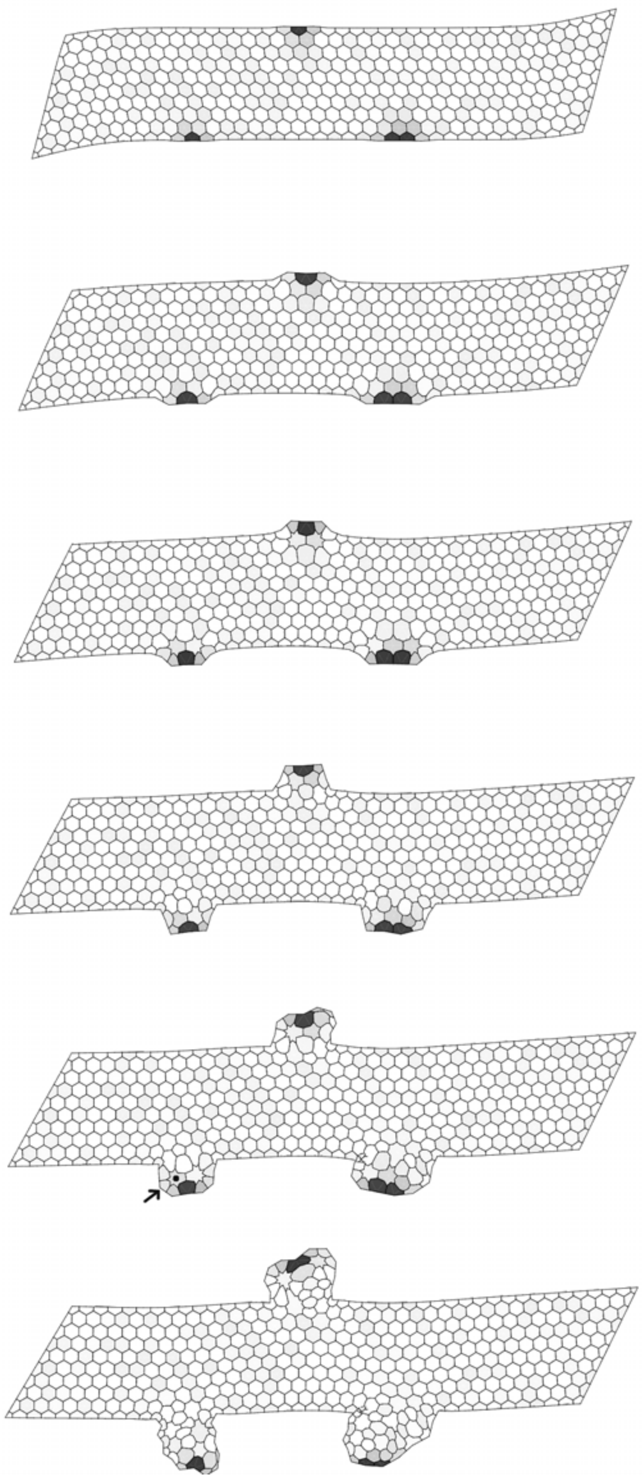


Figure 5.10: A failed experiment in limb growth. Incorrect parameters lead to tumour-like growths. The star polygons (one of which is marked in the second lowest image) are cells that have a high number of neighbours — this should be avoided where possible. The rate of proliferation of the cells on the corners between the limb and body differs. This causes the limbs to curl around. Note that this prototype has no collision detection, so the mesh elements begin to intersect at the corners of the limbs.

5.2 A Study of Drosophila Segmentation

The limb bud model relies on cell communication via morphogen flow in the s-morph. The morphogen gradient can be considered as a one-way communication channel between the AER and the proliferating cells, and the feedback loop enables a two-way communication. In general, dynamic morphogen patterns in SDS can be used to coordinate cell behaviour allowing macro-level structures to be constructed. In theory these patterns can include gradients, stripes, and spots for instance, although further research is needed for more complex patterns (§9.5). This section presents a model of stripe formation based on early segmentation of *Drosophila melanogaster* (fruit fly), and demonstrates the basic patterning capabilities of SDS.

5.2.1 Drosophila Segmentation

The process of partitioning a developing embryo into parallel bands is known as segmentation. *Drosophila melanogaster* is well studied in developmental biology, and many models have been proposed that explain various developmental stages, such as the early segmentation of the egg (Ball, 2001, p101). This section examines a simplified model of Drosophila segmentation restricted to antero-posterior segmentation and the expression of a single band, in order to adapt the principles into SDS.

Drosophila patterning occurs within a *multinucleate syncytial blastoderm* – a complex of cells without the cell membranes that eventually becomes a multicellular complex (Gilbert, 2006, p259). The segmentation process begins with the insertion of maternal factors (mRNA and proteins) into the anterior aspect of the egg (Figure 5.11a). The maternal factors include *bicoid* mRNA which is responsible for antero-posterior axis formation. After translation of the mRNA the *Bicoid* protein diffuses and degrades, eventually forming a morphogen gradient along the antero-posterior axis (Figures 5.11b and 5.11c).

The Bicoid protein activates the *hunchback* gene in the anterior of the egg which produces *Hunchback* protein. The diffusion and degradation of this protein sets up a hunchback antero-posterior concentration gradient. The *Krüppel* gene has two thresholds: a Hunchback concentration that activates it, and a Hunchback concentration that represses it. If the Hunchback concentration is between these two thresholds the gene is activated (Figures 5.11d and e)).

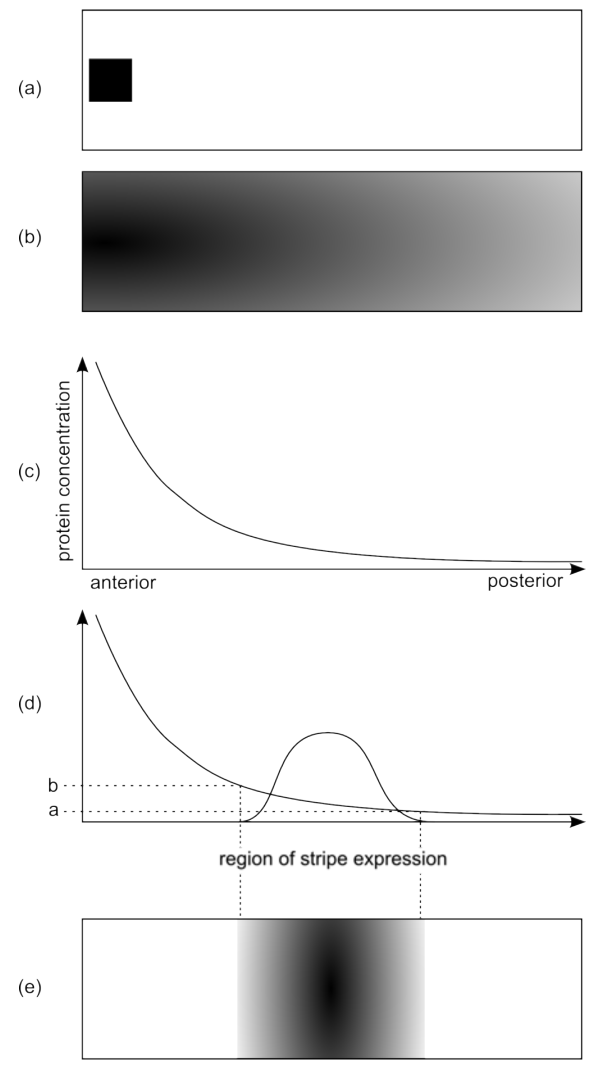


Figure 5.11: An abstract model of Drosphila segmentation. (a) An abstract Drosophila embryo with maternal factors (shaded) injected in one end. (b,c) After a period of time diffusion of the factors have set up a protein gradient. (d,e) The activation threshold, a, and inhibition threshold, b, cause a striped region of genes to be expressed.

5.2.2 Stripe Formation in SDS2

The model of Drosophila segmentation described above uses two main mechanisms to form a stripe: activation and inhibition thresholds, and a morphogen gradient. Both of these can be modelled in SDS: activation and inhibition thresholds can be modelled by rules in the cell program, and, as demonstrated in the limb growth model (§5.1.3.1), a morphogen gradient can be generated by releasing a diffusing and decaying morphogen from a cell. However, the morphogen gradient in Drosophila segmentation is used differently to the gradient used in the limb bud model, time now plays a key role. In order to understand the difference, consider the following hypothetical "first experiment" in stripe formation: Set up an empty s-morph that contains a single special cell that constantly produces a morphogen, ϕ_1 (Figure

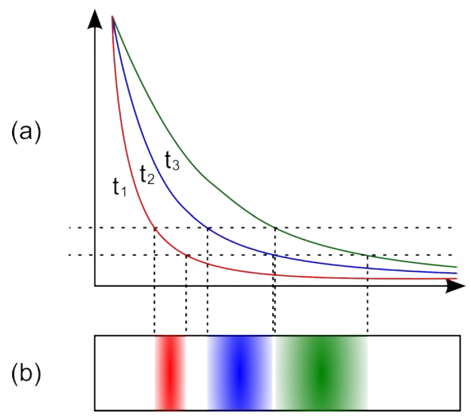


Figure 5.12: A hypothetical morphogen gradient in a simple structure (Figure 5.11a) considered at three increasing times, t_1, t_2, and t_3. (a) The gradient changes over time as the morphogen diffuses further throughout the structure. If genes are activated within a certain concentration band, then (b) the activation regions will change over time also.

5.11a). The morphogen will diffuse and decay, setting up a gradient (Figure 5.11b). However, the gradient in Figure 5.11b is just a snapshot in time of a dynamic process — the distribution of morphogen, and hence activation region, actually change over time (Figure 5.12). In summary, the position and size of the morphogen stripe is determined, not only by the activation and inhibition thresholds and properties of the morphogen, but also by time. Therefore, if a specific position and size of a stripe is required, extra developmental scaffolding is needed.

Taking time into consideration, a stripe generation program was designed for SDS. The program shown in Table 5.2 generates a single morphogen stripe within an s-morph that is used to activate a band of growth. The growth model uses two morphogens, ϕ_1 and ϕ_2. ϕ_1 generates the gradient against which the stripe is perpendicularly aligned. ϕ_2 acts as a *timer* used to control *when* the stripe is expressed, but doesn't diffuse or decay (i.e., $D_{\phi_2} = 0$ and $C_{\phi_2} = 0$). Each cell has a type, which is either *normal*, *injected* (I), or *growing* (G). An injected cell acts as the entry point for the maternal factors modelled by constantly producing morphogen ϕ_1 (rule 1). ϕ_1 activates a slowly changing region of the s-morph, stimulating the production of ϕ_2 (rule 2). Then when a threshold of ϕ_2 is reached, the cell types are changed to *growing* (rule 3). Figure 5.13 illustrates this process (also see animation SDS2/8).

This model has a number of parameters: diffusion and degradation rate for each protein, activation and repression thresholds, rate of growth, and rate of morphogen synthesis. These parameters affect the size, position, and time of activation of the stripe. The parameters used in the simulation of Figure 5.13 were found by trial and error, essentially resorting to a balancing act using a real-time simulator and monitoring the distribution of morphogens over time. This process is tedious due to the large number of parameters; moreover, visual features of the morphogen distribution often depend on multiple parameters. An important area of future research would be to reduce models, such as this one, to a small set of visually relevant parameters. This is discussed in Section 9.6.

Table 5.2: A stripe generation model.

rule	condition	action
r_1:	$c_t = I$	$c_{\phi_1} = \text{vol}(c)$
r_2:	$c_{\phi_1} > a$ & $c_{\phi_1} < b$	$c_{\Delta\phi_2} = \triangle\Delta\phi_2$
r_3:	$c_{\phi_2} > G_{thres}$	$c_t = G$
r_4:	$c_t = G$	$c_{\Delta r} = \triangle r$

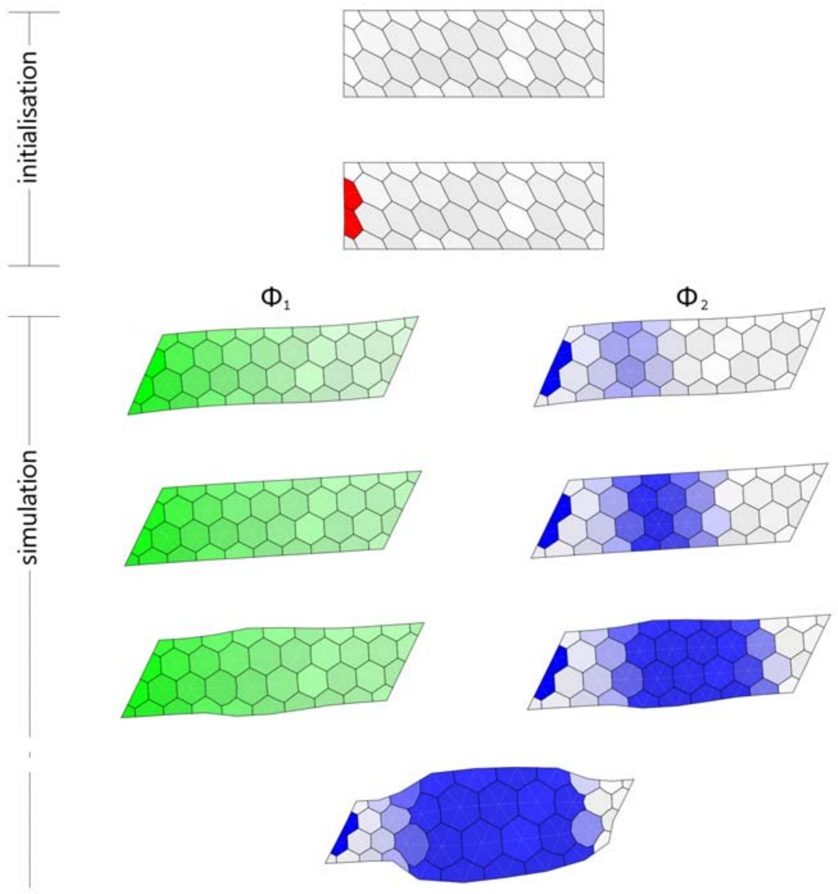


Figure 5.13: (top to bottom) A simulated sequence of Drosophila-like segmentation in SDS2. Initially cells are selected as injected cells (red). The diagrams shows the dual mesh of the s-morph with morphogen concentrations (ϕ_1 is shaded green, and ϕ_2 blue). Note the formation of the band or stripe of the ϕ_2 morphogen. In this simulation ϕ_2 is expressed as cell growth.

5.3 Conclusion

The limb bud experiments demonstrate how directed proliferation, local stimulation using morphogen gradients and activation thresholds, and a feedback loop, can generate limb-like geometries in SDS. While these processes are specified at a cell-level, macro-level structures are formed. Indirect control over the form is gained through a set of parameters which directly affect the generative processes and physical model. The stripe formation experiment illuminates how spatial information can be inferred in an s-morph. By using morphogen diffusion and decay, and activation and inhibition thresholds, we can identify and stimulate a band of cells that lie at a distance from a *coordinator* cell, and form a band perpendicular to the gradient. While this model was not explored further, it offers insight into what other organisational processes could be incorporated within SDS.

It is interesting to consider whether the SDS2 software could have applications in biological research. SDS2 provides an abstraction of a 2D cell layer, and is capable of modelling a number of biological and physical processes. Moreover, as shown in this chapter, it has successfully been used to implement biological models of growth, generating results predicted by the models. Could SDS2 be used as a tool to test hypotheses about biological growth? In its current form, this is doubtful. The components of SDS2 were modelled phenomenologically — the goal being to reproduce the rough behaviour of a system of cells in an efficient manner. Therefore the models in SDS do not accurately represent reality; the physical forces that occur between two adhered cells, for example, are far more complex than those in a single idealised spring. However, some components of SDS, in particular the adaptive mesh representation and the algorithms for cell division and movement, could form the basis of a more sophisticated simulation system appropriate for more accurate biological simulation.

The SDS2 software was built to test and design aspects of SDS, and the experiments presented in this chapter acted as a proof of concept that SDS was capable of producing complex organic structures. Once these initial investigations were completed, research into a 3D implementation of SDS was started. This research is presented in the next chapter.

Chapter 6

A 3D Developmental Modelling System

This chapter presents the details of a 3D implementation of SDS, called SDS3. SDS3 is a system for generating complex organic 3D forms by simulating the biological and physical processes of development in three dimensions. The intended application of SDS3 is in computer graphics, where it could be used to design and autonomously generate complex organic forms for use in computer animation, simulation, games, and design. Existing developmental systems that produce organic forms in 3D typically use a surface representation (Smith, 2006; Combaz and Neyret, 2002; Kaandorp and Kübler, 2001) which fails to capture aspects of internal growth and volumetric material effects. SDS3 addresses this limitation by using a flexible volumetric representation, and is the first developmental system, to my knowledge, that operates on a spatially and structurally dynamic tetrahedral mesh. The research presented in this chapter is a first step towards a method for full biological and physical simulation for 3D modelling, and the results are very promising (Chapter 7).

SDS3 can be considered as a 3D generalisation of SDS2 (Chapter 4), and, in fact, the growth models designed for SDS2 (§5.1.2) can be applied almost directly to SDS3 (§7.1). However, considering it as a generalisation belies both the significant difficulty involved in designing the system, and the fundamental differences between the systems. In 3D, an s-morph is a collection of spherical cells connected together by a tetrahedral mesh. The generalisation from triangles in SDS2 to tetrahedra in SDS3 is by no means straightforward and adds significant conceptual and computational complexity to the implementation. Numerous theoretical and technical challenges were encountered during the implementation of SDS in 3D. This chapter discusses these problems and presents solutions, including an examination of:

- modelling an s-morph as a soft-body in 3D with collision detection between surface elements,
- a simple scheme for rewinding a simulation (in order to perform structural cell movements),
- issues surrounding the conversion of a 3D surface to an s-morph,
- different algorithms for modelling cell division on a tetrahedral mesh,
- an algorithm for performing structural cell movement, and
- a model of morphogen transport on a tetrahedral mesh.

The concepts behind SDS3 developed from the consideration of: SDS2 (Chapter §4), physical simulation techniques, in particular Teschner's tetrahedral mesh approach to deformable modelling (§6.1.1), and existing surface-based developmental systems, such as Vertex-Vertex systems (§6.1.2.3). Before presenting the implementation of SDS3 it is important to consider its historical context in physical and developmental simulation. This is discussed next.

6.1 Related Work

The research presented in this thesis was seeded by imagining the benefits of combining a soft-body physics model with a developmental system. The physical model used in SDS was formulated by considering the latest advancements in 3D deformable modelling (Teschner et al., 2004; Irving et al., 2007). The tetrahedral mesh-based approach of Teschner et al. (2004) was particularly remarkable due to its simplicity and efficiency, and thus became the basis of the SDS3 physical model (§6.3). Teschner's model and various other approaches for physically modelling deformable objects are reviewed in Section 6.1.1. Section 6.1.2 then examines some developmental systems capable of producing organic 3D geometry. These systems have already been introduced (§2); however Section 6.1.2 examines them in more detail, discussing how these systems model developmental events on 3D structures, how they incorporate a physical model, and their relationship with the SDS3 research.

6.1.1 Physical Modelling

The modelling of soft matter in 3D (also known as soft-body or deformable modelling) has been an active area of research in computer graphics for the last two decades (Terzopoulos and Fleischer, 1988). Physical modelling in computer graphics, as opposed to engineering or computational biology, is driven by visual realism and efficiency — if it *looks* real then it's generally good enough. The field of physical modelling in graphics is broad and there are numerous modelling techniques. For a detailed introduction to the field, the following literature is recommended: the survey of Nealen et al. (2005, 2006), the lecture notes of Müller et al. (2008), and the lecture notes of Baraff and Witkin (1997). This section briefly describes some common techniques for modelling deformable objects in 3D.

Mass-spring models have been claimed to be the simplest and most intuitive of all deformable models (Nealen et al., 2005, §3.5). A mass-spring model consists of a finite set of point-particles connected together by springs. The point-particles are infinitely small points in space with mass and velocity. The particles move around in response to the springs and external forces (e.g., gravity). The spring forces are typically modelled by Hooke's law of elasticity, which assumes a linear relationship between elongation and force. Mathematically, the force applied to a particle, a, attached to a particle, b, by a spring with rest length, l, and stiffness, k, is:

$$f(a) = -k\frac{b_x - a_x}{|b_x - a_x|}(|b_x - a_x| - l) \tag{6.1}$$

The benefits of mass-spring systems are that they are intuitive, easy to implement, and efficient. They do not necessarily accurately model any real world material, however this is not so important in animation (as compared to e.g., surgery simulation). A significant issue is that the behaviour of mass-spring systems does not converge under mesh refinement — their behaviour is dependent on the specific resolution and topology of the mesh. This makes it difficult to match model parameters with dynamical behaviour, particularly if the mesh is adaptive, as is the case with SDS. A more detailed discussion of this model can be found in (Müller et al., 2008; Baraff and Witkin, 1997).

Addressing some of the issues of the basic mass-spring model, Teschner et al. (2004) proposed a slightly different model which formed the core of their framework for real-time simulation of 3D deformable objects. Their system models volume-preserving tetrahedral meshes and provides fast methods for detecting and handling colliding surfaces. It is meant for real-time simulation in computer graphics, trading accuracy for efficiency while maintaining visually acceptable results. The physics model in SDS3 is based upon this system, and so it will be discussed in more detail later (§6.3).

The finite element method (FEM) is a more sophisticated modelling technique based on the theory of continuum mechanics. FEM originated from computational engineering but has been adopted into computer animation because of the realistic visual results it can achieve (see e.g., Müller et al., 2008; Baraff and Witkin, 1997). The basic idea behind FEM is to discretise a continuous volume (or shell) into a tetrahedral or hexahedral mesh. The continuum mechanics equations for the volume are then solved at the nodes of the mesh. The nodes store a simple functional description (or basis) of the local distribution of properties (such as stress and strain). Being a physically derived model, FEM provides a good approximation of the mechanics in a solid material and hence the results are quite accurate (much more so than mass-spring models). This accuracy comes at the cost of being harder to understand and more computationally demanding. Recent FEM-based models in computer graphics produce great visual results of soft deformable bodies (Irving et al., 2007). The related method of Finite Differences works similarly but instead discretises the volume into uniform cubes (see e.g., Gibson and Mirtich, 1997; Gibson, 1997).

Mesh-free systems do away with the structural elements that are used by mass-spring models and FEM. Loosely coupled particle systems (Reeves, 1983) model an object as a set of freely moving particles (see also Nealen et al., 2005, §4.1). The particles can be coupled spatially in order to achieve some material cohesion. One coupling approach (Tonnesen, 1992) is to specify an interaction energy between pairs of particles, e.g., $\phi(a, b)$ for all particles a and b. A potential energy for a particle p is then specified as the sum of all the relevant pair-wise energies, e.g., $\phi(p) = \sum_{q \neq p} \phi(p, q)$. The goal is then to minimise the energy of the system. This can be done by specifying a force term $f = -\nabla\phi(p)$ and relating it to particle movement using the typical Newtonian law $f = \ddot{p}m$. These systems can be visualised by an iso-surface in which the particles act as potentials (Blinn, 1982). More sophisticated mesh-free methods known as point-based methods scatter particles over a surface or volume (Nealen et al., 2005, §4.3.1). The continuum elasticity equations are then solved around these points, and as they move they drag the mesh along with them. The advantage of this approach is that the complexity of the mesh is independent of the complexity of the simulation. This approach is gaining popularity with the proposal of a number of recent methods based on these ideas (see e.g., Sifakis et al., 2007; Müller et al., 2005, 2007). These methods are very sophisticated and could be used in conjunction with a developmental model; however, in SDS, a simpler approach is taken, one in which the mesh of an s-morph is used directly.

There are an increasing number of libraries and tool-kits available for simulating deformable bodies, such as Idolib[1], VCG[2], and PhysX [3]. These methods are applied in games, 3D modelling, films, and the relatively young field of surgery simulation. This research began when these libraries were still in their infancy, and at that stage were not quite suited to SDS — in particular SDS uses meshes with a dynamic structure, which some of these libraries do not support. Therefore, considerable time was spent implementing a custom simulator, for both SDS2 and SDS3 (see §8.6.3).

6.1.2 3D Developmental Modelling

Chapter 2 reviewed a number of developmental systems capable of generating complex 3D organic structures. The systems which are most closely related to SDS3 are those that use a 3D mesh representation of a biological structure in conjunction with a model of physical processes. There are a number of ways development can be modelled using such as representation. Kaandorp and Kübler, for example, use a triangulated mesh representation, on which a biological growth "step" is modelled by constructing a new mesh around the old one (§6.1.2.1). The growth is controlled by nutrients distributed through an environment via a physical simulation. Combaz and Neyret, on the other hand, model the physical properties of an elastic shell, which grows depending on the presence of hot-spots painted on the mesh (§6.1.2.2). They use a polygonal mesh to represent the structure, which is automatically refined when more detail is required. The research that is perhaps most related to SDS3 is the Vertex-Vertex system (§6.1.2.3), which can be used to model development acting on a polygonal surface-mesh. Like SDS, cells in a Vertex-Vertex system can be modelled as vertices in a mesh, and as point-masses in a mass-spring system; however, Vertex-Vertex systems are restricted to modelling structures at the surface level, whereas SDS can model volumes. This section examines some aspects of these systems in detail, in order to give historical context to the design of SDS3.

6.1.2.1 Modelling Accretive Growth

In their book *The Algorithmic Beauty of Seaweeds, Sponges, and Corals*, Kaandorp and Kübler (2001) present a variety of sophisticated simulation models of the growth and development of the organisms mentioned in the title. The book offers a number of approaches to modelling the growth of these organisms in submerged

[1]`http://idolib.sourceforge.net/`
[2]`http://vcg.sourceforge.net/`
[3]`http://www.nvidia.com/object/physx_new.html`

environments. Whereas Kaandorp and Kübler are interested in studying biology, their experiments are considered here purely for their 3D form generation potential. An overview of their approach was given in Section 2.2.2.

Kaandorp and Kübler's model of surface normal accretive growth (Kaandorp and Kübler, 2001, §4.6.2) is particularly relevant to this thesis because (a) the modelled organism is represented by a triangular surface mesh, and (b) the simulations result in smooth, organic forms. In this model, a biological entity (representing a colony of organisms) is modelled with a triangular surface mesh embedded in an environment. This model does not have a concept of a *cell*, but rather considers the surface as a continuous growing entity. The organism being modelled grows through the repeated deposition of new *skeleton* material on top of an existing structure (Kaandorp and Kübler, 2001, p125). Growth is modelled by repeatedly applying a geometric operation that constructs a new, larger mesh around an old mesh. By seeding the environment with a simple mesh (e.g., a sphere), this growth process can generate familiar, coral-like forms.

The distance between two meshes, from an old layer j to a new layer $j + 1$, is determined by a growth function, $G()$. A "simple" example of a growth function is:

$$G() = sf(\alpha, \beta)h_2(\ldots) \tag{6.2}$$

where f measures the deviation of the surface normal α from a growth axis, β specifies the leniency of the measure, and $h_2(\ldots)$ approximates the amount of contact the surface has with the environment using its local curvature. This growth function results in the generation of smooth, space filling, forms (Kaandorp and Kübler, 2001, §4.6.2). Kaandorp and Kübler also model a more complex $G()$ term which involves a full hydrodynamical simulation that distributes food particles around the environment (Kaandorp and Kübler, 2001, §4.6.4).

In their model of development, all the complex processes of the development of these organisms have been abstracted away into a single equation which is calculated for each triangle in the mesh. No information affecting the development of the structure is stored in the elements of the organism, but rather, information is gathered at the moment of growth (such as the presence of nutrients). Compare this to SDS, in which cells contain a persistent state which drives the development of the system. As discussed later, in relation to Combaz and Neyret's system (§6.1.2.2), the lack of cellular logic in Kaandorp and Kübler's system makes it unsuitable for modelling a number of interesting developmental phenomenon, such as the development of different types of structures within the same system.

Kaandorp and Kübler models are oriented towards studying morphogenesis of very specific organisms; however, they also happen to produce interesting geometric models with smooth polygonal surfaces, and so are interesting from a computer graphics perspective. This system is unable to satisfy key objectives of this thesis: movement and deformation of form (§1.2); however it does provide an interesting approach to form generation which, with further development, could have wide application in computer graphics.

6.1.2.2 Semi-Interactive Morphogenesis

Combaz and Neyret's semi-interactive morphogenesis system (introduced in §2.2.4) models the growth of a 3D elastic polygonal surface using a *quasi-static* physical simulation, in which every time step is solved to static equilibrium. Their system generates interesting organic shapes that "fold, bend, and curl as in nature" (Combaz and Neyret, 2006). The growth of the surface is induced by hot-spots which a user paints onto the mesh.

One defining feature of Combaz and Neyret's system is their model of an elastic material, which results in the surface contorting and folding as it grows. It is useful to consider their physical model in greater detail, as a point of comparison against the model used by SDS. In Combaz and Neyret's surface growth system, a triangular mesh has a *real* state (i.e., the current vertex positions, edge lengths, and face topology) and a *reference* state, which is specified by desired lengths of each edge, and desired mean curvature of each vertex. The real mesh is updated iteratively by a solver, which minimises the difference between the real and reference states, measured by an energy term, E. This is similar in spirit to the model used in SDS3 (§6.3), which specifies target edge lengths and tetrahedra volumes that the s-morph attempts to achieve. Combaz and Neyret, however, use a more sophisticated energy equation, composed of three parts:

$$E = E_{membrane} + E_{bending} + E_{pressure} \tag{6.3}$$

The membrane and bending energies, $E_{membrane}$ and $E_{bending}$, are derived from the theory of thin shells (Grinspun et al., 2003) and the ad hoc pressure term, $E_{pressure}$, is used to control the folds and curves in the resulting geometry. Combaz and Neyret do not use a volumetric representation, and consequently the pressure term is only a rough approximation to the complex forces that occur within a solid body. In comparison, the use of a tetrahedral mesh representation in SDS supports a more accurate model of internal pressure.

The technical details of the energy calculation are outside the scope of this discussion; however, the complexity of this model can be appreciated by considering, for example, the membrane energy which measures the amount of stretch and shear on a surface. The total membrane energy is computed from the weighted sum of the membrane energies of every triangle, t, in the mesh, as:

$$E_{membrane} = \sum_{t} \bar{A}_t {E^t}_{membrane} \tag{6.4}$$

$$\text{where } {E^t}_{membrane} = \frac{1}{2} \sum_{i=1}^{2} \sum_{j=1}^{2} \sigma_{ij}^{t} \varepsilon_{ij}^{t} \tag{6.5}$$

$\bar{A}_t$ represents the reference (desired) area of t, ε_{ij}^t is the strain tensor (the deformation of t from its reference state), and σ_{ij}^t is the stress tensor (see e.g., (O'Brien and Hodgins, 1999, Eq. 7)). After a *growth step*, which updates the reference state, the simulator minimises the total energy of the system using Conjugate Gradient Optimisation. Each step involves finding the minimal energy state, or *static equilibrium*, of the system. Combaz and Neyret call this method of simulation *quasi-static*.

The quasi-static nature of the physical simulator in Combaz and Neyret's system produces a developmental sequence of forms, each one simulated to static equilibrium. Realistic dynamical behaviour, such as a bouncing ball, is impossible under this scheme (because a ball suspended in mid-air is not in equilibrium); however, this is not an important factor if the desired output of the system is a single geometric model (rather than a continuous animation). The physical model in SDS, on the other hand, is able to generate realistic animations of growing soft-bodied forms, as it has a concept of dynamic properties, such as cell velocity and acceleration. The potential of using an alternate, simpler physical model in SDS is discussed in §8.3.

An interesting feature of Combaz and Neyret's system is that the growth texture (that the user paints) is not stored as a normal 2D image (as this would require constant re-parameterisation of the surface). Instead, the texture information is stored as local information in the vertices and in extra nodes within the faces. Adding extra nodes to the faces allows the resolution of the textural detail to be much greater than the resolution of the mesh. A drawback of the morphogen system used in SDS (§6.6) is that the morphogen simulation is constrained to the resolution of the tetrahedral mesh. It would be desirable, in some situations, to have a higher resolution for the morphogen component, especially if complex patterns are to be generated. Using an approach similar to Combaz and Neyret's (i.e., adding extra *morphogen nodes* to the simplexes), could help achieve this goal.

Combaz and Neyret's system elegantly demonstrates that surface growth combined with an elastic physical model can result in the generation of interesting, organic, curved 3D forms. However, like other systems which model development with a continuum of cells (§2.2.5), it is not clear how to model more complex structures which depend on cells having internal processes. The development of biological structures that rely on sequences of events or complex cell lineages cannot be easily modelled without a concept of cell logic. This observation contributed to the design of SDS as a cell-based system, one in which cells can act out complex programs in order to coordinate the development of different structures.

6.1.2.3 Vertex-Vertex Systems

Vertex-Vertex (VV) systems (introduced in §2.2.3) provide a language for manipulating polygonal surface meshes in 3D. VV systems can be used in a number of ways to model development. For example, a biological cell can be modelled in a VV system either as an individual entity, or as part of a continuum of cells. VV systems can model cells as vertices in a mesh acted upon by physical forces, and in this respect, VV is very similar to SDS. Alternatively, individual cells can be modelled as polygons within a planar map (Smith, 2006, §7.3), and in this respect they are similar to Map L-systems (Lindenmayer and Rozenberg, 1979).

When considering a cell as part of a continuum, a growing tissue can be modelled with mesh operations that abstract larger scale developmental events. This abstraction is demonstrated by Smith in a model of the growth of a primitive branching plant form[4] (Smith, 2006, Chapter 7). Smith models the growth of a mesh by maintaining different sets of vertices, such as an "active ring" which sits at the top of the stem and is responsible for extending the stem upwards by dividing when appropriate. To identify the cells that form the start of new primordia, a morphogen-based model of inhibition–diffusion is simulated that results in a spiral arrangement of new primordia (Smith, 2006, §9.2.2). Surface growth in this model is performed by two specialised functions. The first function models the effect of the directed proliferation of cells within the stimulation zone, by adding a new active ring of vertices to the top of the growing stem. The second function models growth around the primordia, by identifying and subdividing a geometric band near to a specified vertex. These functions are very specific to the model and geometric conditions. Nonetheless, this experiment illustrates how ad hoc models of development can be constructed to validate biological models of growth.

[4]Technically, a shoot apical meristem with growing primordia distributed via a model of phyllotaxis.

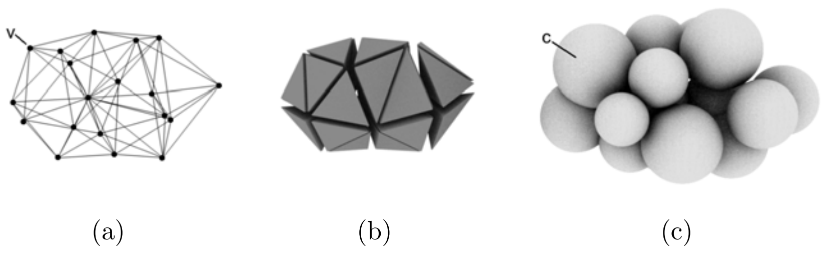


Figure 6.1: (Repeated from Figure 3.3) Three different views of an SDS3 s-morph. It has (a) vertices, edges and (b) tetrahedra, and (c) spherical cells. Vertex v corresponds to cell c.

One disadvantage of the VV method, is that, unlike SDS3, it cannot operate on a volumetric 3D structure. This is because the vertices in a Vertex-Vertex system must have neighbours that can be ordered in a cyclic list. A triangular mesh satisfies this property because every vertex has a set of neighbouring vertices which can be ordered clockwise around it (forming a triangle fan). However, there is no such ordering for tetrahedral meshes, and thus the "algebraic" approach used by VV cannot be applied to tetrahedral meshes.

There is a broad range of approaches to physical and developmental modelling in 3D, each advantageous in specific scenarios or suited to a particular representation. SDS3 is similar to these systems, in that it can produce complex organic 3D surfaces, but differs in one significant respect: it acts on a tetrahedral mesh representation. This representation allows a larger range of developmental and physical phenomenon to be modelled, but poses significant problems. The design of SDS3, that addresses these problems, is described in the remainder of this chapter.

6.2 Overview

This chapter presents the details of a 3D implementation of SDS, called SDS3. In SDS3, an s-morph is composed of spherical cells bound together by a tetrahedral mesh (Figure 6.1). An s-morph in an SDS3 simulation transforms over time, as developmental events and physical effects act upon it (e.g., Figure 6.2). As in SDS2, a valid s-morph in SDS3 is one in which all simplexes are connected. Figure 6.3 shows some examples of valid and invalid meshes — it is assumed that an s-morph is everywhere at least one tetrahedron thick, and so hinges are not allowed.

SDS3 is a complex system that implements the full SDS framework (Algorithm 7). The major components of this system are discussed in the remainder of this chapter. These are:

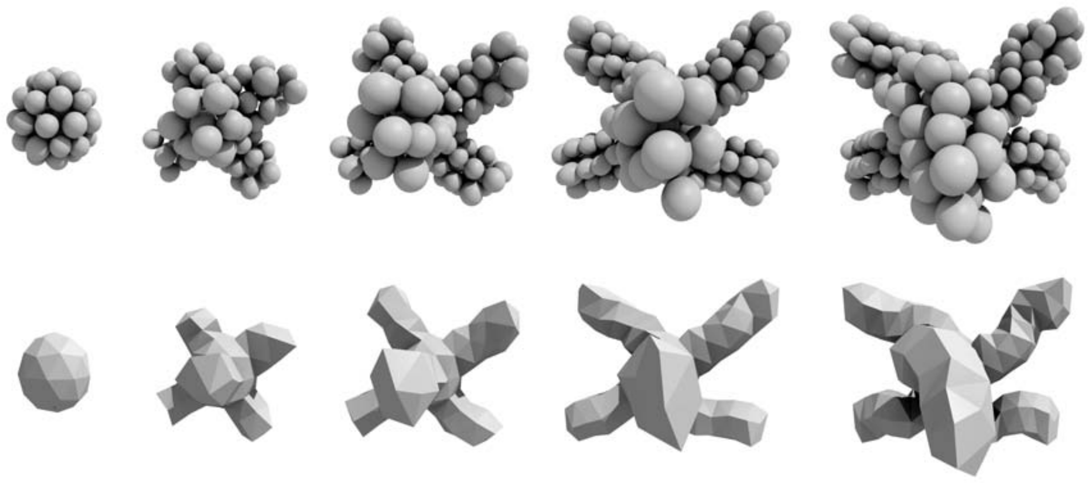


Figure 6.2: (left to right) Frames from the simulation of a developing SDS3 s-morph, shown as (top row) cells and (bottom row) the surface of the tetrahedral mesh.

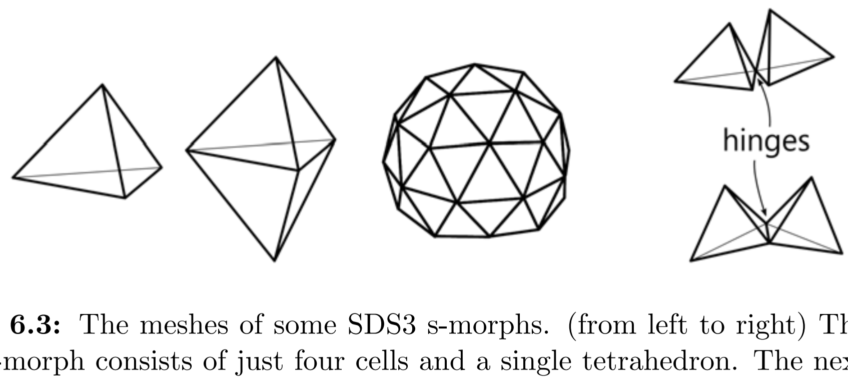


Figure 6.3: The meshes of some SDS3 s-morphs. (from left to right) The simplest SDS3 s-morph consists of just four cells and a single tetrahedron. The next simplest consists of two tetrahedra joined at a face. The third image shows the surface mesh of a complex ellipsoid s-morph. Hinges, such as those shown on the right, are not allowed.

- The physical model and related issues, such as collision detection and adaptive time-stepping (§6.3),
- A comparison of algorithms for modelling cell division in a tetrahedral mesh (§6.4),
- An algorithm for modelling cell movement within a tetrahedral mesh (§6.5), and
- The discretisation of the morphogen model on a tetrahedral mesh (§6.6).

Algorithm 7 SDS3 Simulation

```
$t = 0$
while $t < duration$ do
  SimulatePhysics($\Delta t$)
  if MovementDetected() then
    $\widetilde{\Delta t}$ = TimeOfFirstMovement()
    RewindSimulation($\widetilde{\Delta t}$)
    PerformCellMove()
  else
    $\widetilde{\Delta t} = \Delta t$
  end if
  HandleCollisions()
  UpdateCellState($\widetilde{\Delta t}$)
  SimulateMorphogens($\widetilde{\Delta t}$)
  RunCellPrograms($\widetilde{\Delta t}$) (may call PerformCellDivide)
  $t = t + \widetilde{\Delta t}$
  Output world state for time $t$
end while
```

6.3 Physical Model

The physical model in SDS3 generalises the concepts of the SDS2 physical model. Cells have a position, velocity, size, and mass, and move around in response to forces applied from the simplexes. Whereas the SDS2 model only incorporated springs along edges, SDS3 models both edges and tetrahedra with springs. This approach is based on the framework of Teschner et al., introduced above (§6.1.1). This section describes how this model has been adapted to fit into the SDS framework — in particular how to couple cell size with spring rest size and how to incorporate simulation rewinding through the use of an adaptive simulation time-step. Other key considerations of the physical simulator discussed are the modelling of cell state changes, the concept of a rest scale multiplier, and frozen cells.

Conceptually, the tetrahedral mesh represents a solid deformable body, with each tetrahedron representing a homogenous elastic element. The model provides local volume conservation, is more stable than the basic mass-spring model, and converges to the "true solution" under mesh refinement. Figure 6.4 shows the physical simulation of a spherical s-morph acted upon by gravity and colliding with a plane, demonstrating all of these aspects of the model in action. The physical equations that govern the soft-body dynamics of an s-morph in SDS3 will now be described.

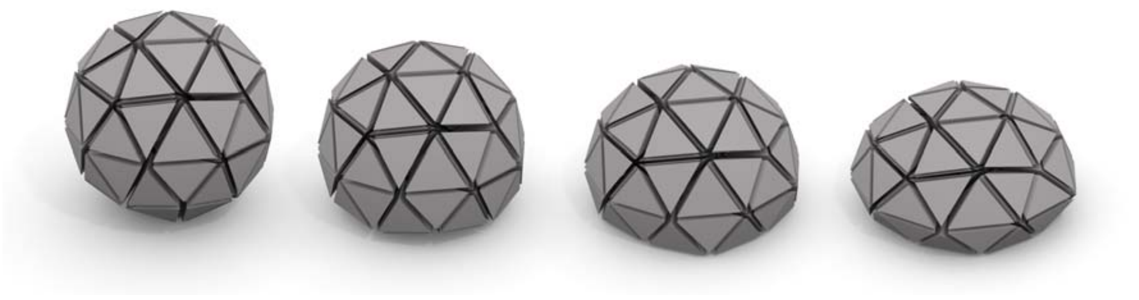


Figure 6.4: (left to right) A spherical s-morph is placed onto a plane and, under gravity, deforms into an ellipsoid. The final stable state occurs when the reaction force from the plane balances against the gravity force.

6.3.1 Constraints and Potential Energy

The physical model is most easily understood with the concept of *constraints*. The simplexes within an s-morph try to maintain a consistent size, and hence constrain the motion of the cells. As outlined in §3.6.2, the cells of an s-morph are modelled as point masses with position, c_x, and mass, c_m. A set of constraints describes an *ideal* configuration of the cells. These constraints exist for every edge and tetrahedron[5] and are *satisfied* when they equal zero. As an example, for an edge, e, joining two cells, a and b, the following constraint specifies that the two cells should be separated by a distance equal to the sum of their radii (see Figure 6.5).

$$C(e) = |a_x - b_x| - (a_r + b_r) \tag{6.6}$$

In practice there are numerous constraints, all of which cannot be satisfied simultaneously. Hence, instead of finding the optimal solution, a mesh configuration that satisfies the constraints as closely as possible is sought. The constraint solver is a key component of Teschner's model, which models the constraints as *potential energies*, $E(s)$. These potential energies are converted into forces, $F_s(c)$, acting on the individual cells. The forces on the cells can be interpreted as the direction the point

[5]In practice, applying constraints to faces is not necessary when using both edge and tetrahedron constraints (Teschner et al., 2004).

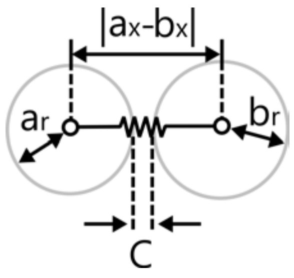


Figure 6.5: For an edge connecting two cells, the constraint equation, C, measures the shortest distance between the boundaries of the two cells. In this example $C > 0$.

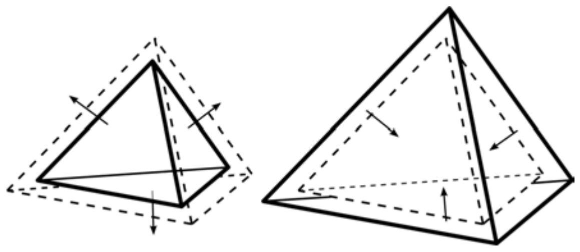


Figure 6.6: The tetrahedral constraints attempt to preserve the volume of the tetrahedra. Two tetrahedra in their current states (solid line) and desired states (dashed line), and the forces acting upon them due to the constraints.

masses need to move in order to reduce the potential energy of the system. The forces and cell movement are related by Newton's law, $F = ma$, to generate a set of equations that describe the movement of cells over time. These piece-wise ordinary differential equations are then numerically solved to obtain the positions of the cells at discrete time steps.

6.3.2 Calculating the Forces

The desired state of an s-morph is specified by a set of constraints. Every edge and tetrahedron has a constraint, the edge constraints preserve edge length and the tetrahedral constraints preserve tetrahedron volume. For every simplex, s, whether it is an edge or tetrahedron, the constraint is given by:

$$C(s) = \frac{V(s) - R(s)}{R(s)}, \tag{6.7}$$

where $V(s)$ is the current size of the simplex and $R(s)$ is the desired, or *rest*, size. For an edge, e, connecting cells a and b:

$$V(e) = |a_x - b_x| \tag{6.8}$$

$$R(e) = a_r + b_r \tag{6.9}$$

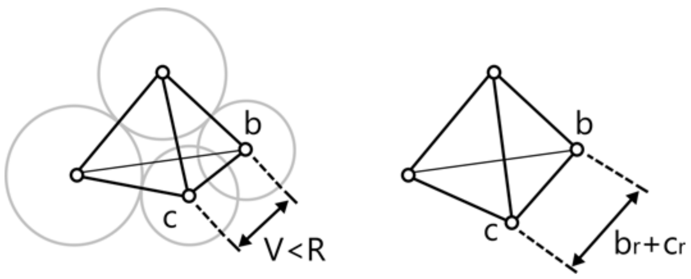


Figure 6.7: Computing the rest size of a tetrahedron. (left) A tetrahedron of an s-morph and its four attached cells. The edge bc has length V which is less than R. (right) Construct a tetrahedron where the length of each edge is the sum of the radii of the attached cells. The desired volume of the original tetrahedron is equal to the volume of this new tetrahedron, which can be calculated using Heron's formula.

The signed volume of a tetrahedron, t, is:

$$\mathrm{vol}(t) = \frac{1}{6}(v_1 - v_0) \cdot ((v_2 - v_0) \times (v_3 - v_0)), \tag{6.10}$$

where the v_i are the positions of the four vertices of the tetrahedron. The rest size of a tetrahedron t with cells a,b,c and d is computed by first calculating the desired lengths of all edges of t (Figure 6.7) and then using Heron's formula to compute the volume of the tetrahedron with those edge lengths:

$$\begin{aligned}
R(t) &= \frac{\sqrt{ABCD}}{3(a_r + d_r)(b_r + d_r)(c_r + d_r)}, \text{ with} \\
A &= w + x + y - z \\
B &= w + x - y + z \\
C &= w - x + y + z \\
D &= -w + x + y + z \\
w &= a_r b_r c_r \\
x &= d_r \sqrt{b_r c_r (a_r + b_r + d_r)(a_r + c_r + d_r)} \\
y &= d_r \sqrt{a_r c_r (a_r + b_r + d_r)(b_r + c_r + d_r)} \\
z &= d_r \sqrt{a_r b_r (a_r + c_r + d_r)(b_r + c_r + d_r)}
\end{aligned} \tag{6.11}$$

As defined in §3.6.2, each simplex, s, has physical properties associated with it, a stiffness coefficient, s_{k_d}, and a damping coefficient, $s_{k_{damp}}$. As noted above, only edges and tetrahedra are actually considered to have these properties as faces are not necessary in the physical model (alternatively, all faces can be considered to have zero stiffness and damping). The constraints are modelled elastically using a potential energy function based on Hooke's law of linear elasticity:

$$E(s) = \frac{1}{2} s_{k_d} \left(\frac{V(s) - R(s)}{R(s)} \right)^2 \tag{6.12}$$

Note that this equation is similar to the SDS2 energy equation (Equation 4.3) except s now denotes either an edge or a tetrahedron, and there is a normalisation factor. This factor makes the spring stiffnesses *scale-free* which allows the same stiffness coefficients to be used throughout the entire structure (Teschner et al., 2004). The spring stiffness and damping coefficients (discussed later) could easily be specified per spring[6]. The actual coefficients vary from experiment to experiment and generally have to be manually fine-tuned to achieve the desired results. The specific values of these parameters will be dependent on the implementation (a side-effect of the complexity of the software).[7]

Given the energy, $E(s)$, of a simplex, s, the force acting on a cell, c, is defined as:

$$F_s(c) = -\frac{\partial}{\partial c_x} E(s) \tag{6.13}$$

The total force on a cell is the sum of all forces from adjacent simplexes:

$$F(c) = \sum_{s:c \in s} F_s(c) \tag{6.14}$$

This force affects the cell position as follows:

$$\frac{d^2 c_x}{dt^2} = \frac{F(c)}{c_m} \tag{6.15}$$

A useful alternative interpretation of this model is to consider two spherical cells joined together by a "sticky" region on their surfaces. The region resists being separated, and thus force is required to separate the two cells. Imagine the cells also contain a fluid, that will resist being compressed, and thus pushing the two cells together will result in an opposing force. These phenomena are not actually modelled in SDS, but the interpretation helps to understand the physical behaviour of the system.

A common technique to improve the robustness of a simulation is to incorporate damping; Moreover, it has been determined that, to increase robustness, this kind

[6]For the results presented in Chapter 7, one set of spring coefficients is used for surface springs and another set is used for internal springs. This can be utilised, for example, to model the difference in physical behaviour of the mesenchymal and epithelial cells in the limb bud model using looser internal springs and tighter surface springs.

[7]For illustrative purposes, the physical parameters of the single curling limb experiment (§7.7) are: $s_{k_d} = 17$, $s_{k_{damp}} = 0.05$, and $V_{damp} = 0.05$.

of system only needs damping on the tetrahedral elements (Teschner et al., 2004). The damping on a tetrahedron is modelled by providing a resistive force proportional to the velocity of the cells. This is incorporated into the full equation force acting on a cell, c, from a simplex, s:

$$F_s(c) = -\left(s_{k_d} C + s_{k_{damp}} \sum_{\tilde{c} \in s} \frac{\partial C(s)}{\partial \tilde{c_x}} \tilde{c_v} \right) \frac{\partial C}{\partial c_x}, \tag{6.16}$$

where C was given in Equation 6.7. Given the current state of the mesh, F_s can be calculated for the edges and tetrahedra. For the sake of completeness the full expansion for F_s is given in Appendix D. Damping of the system is also incorporated as explicit viscous damping of all cells, via the V_{damp} parameter (discussed below).

6.3.3 The Response of Cells to Forces

The dynamical system specified in Equation 6.15 can be solved between two points t and $t+h$ using a number of numerical integration techniques[8] (see e.g., Müller et al. (2008) for a review of integration schemes). The SDS3 implementation employs the Verlet integration scheme as it is fast and offers increased stability over similarly efficient methods (Teschner et al., 2004). Under this scheme, cell positions are modified over a time interval using the following equation:

$$c_x(t+h) = c_x(t) + (1 - V_{damp})(c_x(t) - c_x(t-h) + h^2 \frac{F(c)}{c_m}) \tag{6.17}$$

This equation also incorporates an ad hoc damping factor V_{damp} that represents a viscous fluid in which the s-morph is embedded. The purpose of this constant is to improve the stability of the simulator, but it also provides an additional environmental control parameter of the generative system (the same growth model embedded in different viscosity environments will generate different forms). Implementations can compute the Verlet equation simply by storing the positions of each cell for the current time step, t, and previous time step, $t-h$. However, due to cell movement transformations (§6.5), the last position may not have occurred at $t-h$, but rather at a time $t-h'$, where $h' < h$. Hence an adaptive Verlet integration scheme is used (based on Dummer, 2005) in which the position of the last time step is linearly extrapolated:

$$c_x(t-h) \approx c_x(t-h') \frac{h}{h'} \tag{6.18}$$

[8]Not every integration technique is suitable. The integration technique should assume linear velocity of all cells between the two time points — this is required by the rewinding scheme (§6.3.4).

In the Verlet scheme, the velocity of a cell is not stored explicitly but is computed from the positional data of the current and previous time steps. In the cases where the velocity of a cell is required, as in Equation 6.16, the cell velocity can be calculated as:

$$c_v(t+h) = \frac{c_x(t+h) - c_x(t)}{h} \tag{6.19}$$

6.3.4 Rewinding Time

Structural cell movement in SDS2 occurs when cells cross edges in the triangular mesh (§4.5). This event also occurs in SDS3 (the structural issues are described in Section 6.5). During some interval $[t, t+h]$ many structural cell movements may have occurred. Instead of handling them simultaneously it is simpler to "rewind" the simulation back to the time of the first movement event, perform a single cell movement transformation and continue the simulation from that point onwards[9].

Assuming the simulation is at time $t+h$, the goal of the simulator is to rewind the state of the world to a time $t+h'$, where $0 < h' < h$. One solution to this task would be to remember the state of the s-morph at time t, allowing us to reset the positions of the cells to time t and then step forward by an amount h'. But this requires integrating the physical equations again. A cheaper solution arises by assuming that the cells have linear velocities over the time interval $[t, t+h]$ and then interpolating their positions[10]. In this case the state of each cell at the desired time, $t+h'$, is:

$$c_x(t+h') = \text{lerp}(c_x(t), c_x(t+h), \frac{h'}{h}), \tag{6.20}$$

using the linear interpolation function: $\text{lerp}(a, b, t) = a(1-t) + bt$.

6.3.5 Handling Collisions

The 3D forms that SDS generates do not have intersecting geometric elements. This is made possible with the incorporation of a system that detects and handles collisions between the surface elements of the s-morphs, world boundary, and static geometries within an environment. Within the simulation loop, the physical simulation first updates the positions of all the cells. The purpose of the `HandleCollisions()`

[9]A disadvantage of this approach is that it may slow down a simulation drastically if many cell movements occur. It may be necessary, if massive cell migration is to be modelled for example, to explore alternatives to this approach, such as performing multiple movements in parallel. In the experiments presented in the next chapter, cell movements were rare, and so performing sequential cell movement transformations was not too computationally demanding.

[10]This assumption is correct in the case of Verlet integration, but may be less accurate if more advanced integration techniques are used.

procedure is to ensure that no mesh elements intersect. Collision detection and handling is a very active field of research and there are a number of methods available for handling collisions between deformable bodies (Teschner et al., 2005). SDS3 incorporates the fast and robust collision detection and response framework proposed by Teschner et al. (Teschner et al., 2003; Heidelberger et al., 2004) which operates directly with tetrahedral meshes and integrates nicely with the simulation method. The system efficiently detects when a cell has penetrated a surface face and projects it backwards until it is no longer intersecting. The same system is used to model collisions with static objects, other s-morphs, and to confine the s-morph within the bounds of the environment.

6.3.6 Cell State Changes

Non-spatial cell state changes (those updated with the `UpdateCellState`(Δt) procedure) can also be considered to occur within the physical simulation. The most important state change is cell growth. By changing size, cells affect the lengths of edges and the shape of tetrahedra, allowing the development of different sized regions. Hence one part of an s-morph can be coarsely modelled with large tetrahedra, while simultaneously another part can have many small tetrahedra modelling smaller features.

The cell state changes in SDS3 are updated as in SDS2, using an explicit Euler step. Cell growth, for example, is updated with the following equation:

$$c_r(t+h) = c_r(t) + c_{\Delta r}(t)h, \tag{6.21}$$

where $c_{\Delta r}$ is the linear rate of growth of cell c.

6.3.7 Importing an S-morph

A popular geometric representation used in computer graphics is the triangular surface mesh. This representation is readily supported in most 3D modelling packages and is accelerated by modern graphics hardware. Tetrahedral meshes, on the other hand, are not supported in most 3D modelling packages. Fortunately, as the surface of a tetrahedral mesh is a triangular surface mesh, each representation can be converted into the other. Practically, this means that an SDS3 s-morph can be imported into a 3D modelling package for analysis or rendering. In the same way, the initial conditions of an SDS simulation can be prepared using a 3D modeller.

Converting an SDS3 s-morph to a triangular surface mesh can be accomplished by exporting the surface cells as the vertices of the mesh, and the faces of the tetrahedra that lie on the surface as the triangles. This conversion loses most of the information within the s-morph, but this doesn't matter here. Some of the figures in the next chapter were rendered by importing an SDS3 simulation into a 3D animation package using this method. In addition to this, some other visualisation methods were used: cell visualisation, which converts the cells to sphere primitives and discards all other information in the s-morph; and tetrahedron visualisation, which converts each individual tetrahedron within the s-morph to its own surface mesh, shrinking it slightly. Each visualisation is useful in understanding the results of a simulation. Furthermore, some of the s-morph boundary meshes were smoothed using Catmull-Clark subdivision surfaces, to create a more aesthetically pleasing appearance. [11]

One user-friendly method to set up a simulation is to construct a surface mesh within an interactive 3D modelling program, and then export that mesh as the initial s-morph for an SDS3 simulation. This method was found to be adequate for most cases and was used to set up all of the experiments presented in the next chapter. The conversion from a triangular mesh to a tetrahedral mesh was performed using a library for tetrahedralising piece-wise linear complexes, Tetgen[12]. The SDS3 simulator takes a tetrahedral complex as input, and forms the mesh of an s-morph, from which the cell information is inferred. SDS3 computes the radii (and hence mass) of all cells based on the edge lengths in the mesh:

$$c_r = \frac{1}{2|N(c)|} \sum_{n \in N(c)} |c_x - n_x| \tag{6.22}$$

Using the cell radii, the rest sizes, $R(s)$, can be computed for all edges and tetrahedra. Initially this procedure seemed adequate, however, importing a tetrahedral complex that has non-regular tetrahedra (a common occurrence) resulted in a deformation of the initial shape. The SDS2 experiments show an initial square s-morph deforming into a trapezoid (Figure 5.2), an analogous issue arises in 3D but is much more problematic, particularly when importing smooth meshes, such as spheres.

The *rest scale multiplier* addresses the undesirable deformation that occurs when importing. It is a per simplex variable, s_m, which stores the difference between the initial state of the imported mesh and the desired state as specified by the inferred

[11] In hindsight, an alternative subdivision scheme, such as Loop subdivision (Loop, 1987), would have been more suitable due to the triangulated nature of the surface mesh.

[12] `http://tetgen.berlios.de/`

cell radii. When a mesh is imported, this variable is initialised as:

$$s_m = \frac{V(s)}{R(s)} \tag{6.23}$$

The rest scale multiplier is used in the physical equations to calculate a modified rest size:

$$R'(s) = s_m R(s), \tag{6.24}$$

which is used in place of the rest size, R, in all the energy calculations. This has the effect that the initial minimal energy state of the s-morph corresponds precisely to its initial imported state, which eliminates the deformation at the start of the simulation.

6.3.8 Frozen Cells

It occasionally becomes useful to hold part of an s-morph in place, stopping it from growing and moving due to forces, but still allowing it to take part in physical collisions. For example, if growing a tree-like structure, *freezing* the base as SDS simulates the growth of a branch may increase stability and controllability. This is implemented in some of the SDS3 experiments with a Boolean cell variable, c_{frozen}. When frozen, a cell does not move or grow, but it retains the ability to transport morphogens and physically collide. The implementation is simple: a frozen cell doesn't have its position updated during the integration step (and for efficiency simplex forces are not computed if the simplex is composed entirely of frozen cells.) The frozen cell variable can be modified by the cell program or assigned to a whole region of a mesh when importing from a surface mesh into SDS3. In the latter case, where the value of c_{frozen} is given only for surface cells, it may be necessary to infer which internal cells are frozen. One approach is to propagate the c_{frozen} variable from the surface down into the s-morph, assigning cells as frozen only if the majority of their neighbours are frozen (Algorithm 8 (p122)).

The physical model describes the motion of the cells and results in changes in shape of an s-morph. The next two sections discuss how the structure changes due to cell division and movement.

6.4 Cell Division

Cell division in SDS replaces one cell with two or more cells. At the mesh level, cell division replaces one vertex of the mesh with two or more vertices placed close

Algorithm 8 Frozen cell propagation algorithm.

```
Let $P$ be the set of all surface cells
while $N(P)/P \neq \emptyset$ do
  $Q = \{\}$
  for $c \in N(P)/P$ do
    $M = N(c) \cap P$
    $F = \{m \in M : m_{frozen} = \text{true}\}$
    Let $c_{frozen} = $ true if $|F| > |M|/2$ or false otherwise
    Add $c$ to $Q$
  end for
  $P = P \cup Q$
end while
```

together and aligned along a division direction. In order to maintain the tetrahedralised structure of an s-morph, the configuration of the tetrahedra must be adjusted. Additionally, the changes to the structure should be as local as possible (for reasons discussed in §3.5.2).

Cell division in SDS3 is considerably more complex than in SDS2. It is difficult to even imagine the different arrangements of tetrahedra that can surround a cell and to understand how the configurations can be changed with the removal and addition of vertices. There are many different ways a cell division transformation can be implemented in SDS3, and one aspect of this research was exploration — theoretical and experimental — of the consequences of some different approaches.

In the formative stages of this research, an attempt was made to enumerate all the different configurations of tetrahedra that can surround a single cell, in order to design an efficient and robust division algorithm. This task was extremely complex and time-consuming, and was eventually suspended in favour for the simpler cell division algorithms presented below. The goal of this research was not, after all, to design perfectly efficient algorithms, but to demonstrate the generation of complex organic geometries in SDS, and the simpler cell division algorithms fulfil this goal. Nonetheless, notes on this original line of research are presented in Appendix B, and may offer insight for future developments.

This section examines a number of approaches to cell division. First tetrahedral subdivision is examined along with its generalisation, simplex subdivision. This technique provides a simple and efficient way to model cell division but results in topological asymmetry amongst the daughter cells. A general method is then proposed that uses a black-box tetrahedralisation algorithm. (With a good tetrahedralisation algorithm this method is computationally more expensive than subdivision, but can result in a more balanced distribution of structure around the daughter

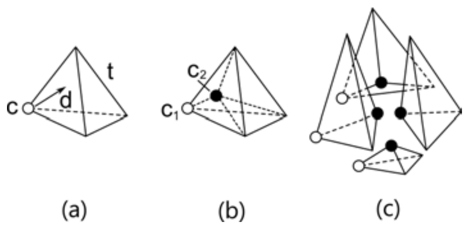


Figure 6.8: Cell division modelled with tetrahedron subdivision. (a) Given a dividing cell c, select the adjacent tetrahedron, t, that lies in the direction of division, d. (b) Split t into four new tetrahedra by adding a new cell to its centroid. (c) An expanded view of t split into the four new tetrahedra, with the location of the first daughter cell (white) and the second daughter cell (black) highlighted.

cells.) The method used for the SDS3 implementation is then described, which uses a combination of simplex-subdivision and the general method.

Tetrahedral Subdivision The concept of modelling cell division in SDS2 by subdividing neighbouring triangles (§4.4.1) can be generalised to SDS3. This procedure is outlined in Algorithm 9 and illustrated in Figure 6.8. Consider an internal cell, c, that chooses to divide in direction, d. Choose the neighbouring tetrahedra, t, in direction d (t exists because the cell is internal). Subdivide t by adding a new cell at its centroid and replacing t with four new tetrahedra. Let the daughter cells consist of the original cell, c, and the new centroid cell.

Algorithm 9 Cell division as tetrahedral subdivision

```
Input: $c \in C$, $d \in \mathbb{R}^3$
Choose the $t \in T$ that lies in direction $d$, where $c \in t$ (Figure 6.8a)
Subdivide $t$ by adding a new cell $c_2$ at the centroid (Figure 6.8b and c)
Rename $c$ to $c_1$
Output: $c_1$ and $c_2$
```

The operation is simple, but it is unbalanced in two ways. Firstly, the local topology around the mother cell is not distributed evenly amongst the daughter cells. Cell c_2 always has four neighbours, whereas cell c_1 has an arbitrary number, which violates the topological symmetry guideline (§3.5.2). Secondly, if the two daughter cells are to be of equal size, the structure is not physically stable, because c_2 is "trapped" inside the subdivided tetrahedra, but it is so big that stresses acting on it will push it out. One way to relieve this is to give the cell c_2 a smaller size than c_1, resulting in both physical and topological asymmetry. Nonetheless, tetrahedral subdivision is very simple to implement, isolates all changes to the hull of a single tetrahedron, and only adds three new tetrahedra to the system. On the other hand, it is asymmetric and restricts the directions in which a cell can divide.

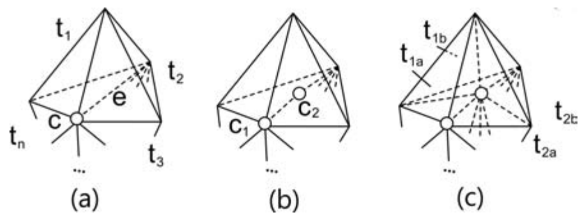


Figure 6.9: Internal cell division by subdividing an edge. (a) Given a cell c, choose an adjacent edge e. Label the tetrahedra attached to e in sequence as $t_1 \dots t_n$. (b) Subdivide e and let the two new daughter cells be c_1 and c_2. (c) What remains is to split each of the t_i into two tetrahedra t_{ia} and t_{ib}, as shown.

Simplex Subdivision We can generalise tetrahedral subdivision to *simplex subdivision*, whereby we allow any simplex that contains c to be subdivided. Any j-simplex can be subdivided into $j+1$ j-simplexes. Tetrahedral subdivision is simply the case where $j = 3$. For the case $j < 3$, subdivision of a j-simplex results in additional topological changes. Figures 6.9 and 6.10 demonstrates edge and face subdivision in a tetrahedral mesh. These operations give a cell a greater range of directions in which it can divide.

Note that this subdivision will always add just one new vertex, however each method adds a different number of tetrahedra. Subdividing faces or tetrahedra only adds a constant number of new tetrahedra (3 and 4 respectively), but subdividing an edge will add n new tetrahedra, where n is the number of tetrahedra attached to the edge.

A more flexible algorithm would choose between these different types of subdivision depending on the direction of division. The first iteration of the SDS3 prototype used a combination of tetrahedral and edge subdivision. If an edge attached to the dividing cell was aligned parallel to the desired direction of division, then it would be subdivided, otherwise the tetrahedron pointed to by the division direction would be subdivided.

Simplex subdivision also applies to division along a boundary. Figure 6.11 illustrates a boundary cell, c, dividing in direction, d. Boundary division can be modelled by subdividing a neighbouring surface face in direction d, which can be interpreted as subdividing the tetrahedron attached to the face into three parts. A similar operation exists for surface edge subdivision that acts just like internal edge subdivision (Figure 6.9), except in this case tetrahedra t_1 and t_2 are not neighbours.

Balanced division approach Tetrahedral subdivision results in structural asymmetry and a reduced fidelity in directional division. Edge and face subdivision are marginally better, but still suffer from asymmetry. An ideal division algorithm

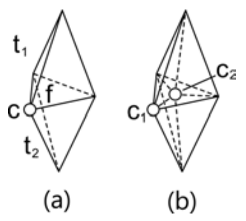


Figure 6.10: Internal cell division by subdividing a face. (a) Given a cell, c, choose an adjacent face, f. As c is internal, the face is connected to two tetrahedra, t_1 and t_2. (b) Subdivide f by adding a new cell, c_2, at its centroid. The daughter cells are c_1 and c_2. Subdividing the face splits t_1 and t_2 into three tetrahedra each.

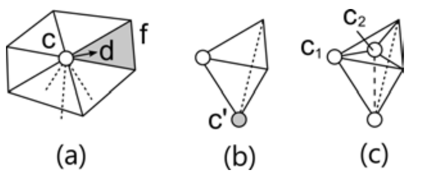


Figure 6.11: Boundary cell division with face subdivision. (a) A boundary cell, c, and its neighbouring surface faces. Given a direction of division, d, choose the surface face, f, (shaded) which lies in that direction. (b) Identify cell c' which lies opposite f. (c) Subdivide f and its adjacent tetrahedra by adding a new cell c_2 to the centroid of f. Rename c to c_1.

would support arbitrary division direction, maintain as much of the local topology as possible, and distribute the topology of the mother cell evenly amongst its daughter cells. Some theoretical work towards such an algorithm is presented in Appendix B. For the results presented in this thesis a "brute-force" approach was taken.

Algorithm 10 presents a general approach to performing cell division. The algorithm removes all elements adjacent to the dividing cell, c, removes c itself, adds the two daughter cells a and b aligned along d, and then tetrahedralises the empty hull with the daughter cells inside. The tetrahedralisation phase can be performed using any appropriate algorithm or library, depending on the quality of the structure that is required. (In the SDS3 implementation a Delaunay tetrahedralisation is generated in this step using Tetgen[13] tetrahedralisation library.)

Algorithm 10 Balanced cell division

```
Let $c$ be the dividing cell, and $d$ be the desired direction of division
Let $T_c$ be the set of all tetrahedra adjacent to $c$
Let $F$ be the hull surrounding $\cup T_c$
Remove $\cup T_c$ and $c$
Add two cells $a$ and $b$, with $a_x = c_x + \epsilon d$ and $b_x = c_x - \epsilon d$, where $\epsilon$ is such that $a$
and $b$ are contained within $F$
Tetrahedralise the structure $F \cup \{a, b\}$
```

[13]`http://tetgen.berlios.de/`

As in Algorithm 2 (p50), the mass of the mother cell c_m is then distributed evenly amongst the daughter cells, and the radii of the new cells are computed from these masses assuming a uniform density. All new tetrahedra and edges then have their rest sizes calculated (§6.3.2). This allows the local volume to be preserved, and existing local stress will be transferred into the new configuration.

Cells that lie on the boundary of the mesh can also be divided using this approach, with a small modification. If a cell c lies on the surface, then, in Algorithm 10, c lies on the boundary of T_c. This means that when $\cup T_c$ and c are removed, some of the faces on the hull, F, will also need to be removed. The structure $F \cup \{a, b\}$, then, is an open hull (i.e., not a closed one as in the internal cell case). This open hull then needs to be tetrahedralised appropriately (e.g., with Tetgen, which can tetrahedralise closed or open hulls). If the tetrahedralisation algorithm cannot operate on open hulls, then the open region may need to be closed first (e.g., by patching it with faces).

Implementation Details The SDS3 implementation used to generate the results shown in the next chapter incorporates a combination of the above mentioned division methods (Algorithm 11). This algorithm does not support division away from the surface, however this operation was not required in the experimentation of SDS3. For division along a surface, the algorithm either performs edge subdivision if the direction of division is oriented within an angle, α, of an adjacent edge, or performs surface face subdivision otherwise (Figure 6.12). For internal cell division, the algorithm either performs edge subdivision if the direction of division is oriented within an angle, α, of an adjacent edge, or performs balanced cell division otherwise. This hybrid approach works because when the direction of division lies almost parallel with an adjacent edge, it is efficient and sufficient (for the experimental results) simply to subdivide the edge, rather than performing a brute-force re-tetrahedralisation.

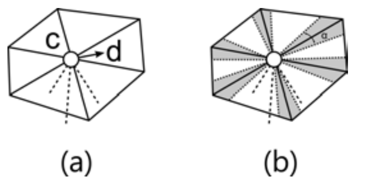


Figure 6.12: Selecting between edge and face subdivision for a surface cell. (a) Given a surface cell, c, with an intended direction of division, d, a choice is made to either subdivide a neighbouring edge or face based on the direction. (b) If the d lies within an angle α of any edge (the shaded regions) then that edge is subdivided. Otherwise the neighbouring face that d is pointing towards is subdivided.

The tetrahedralisation step in Algorithm 10 (p125) is performed using Tetgen. The library generates a Delaunay tetrahedralisation that produces almost regular tetrahedra, and hence the resulting configuration has a low energy. One side-effect of this method is that more than two daughter cells may be generated in order to keep the tetrahedra of a minimum quality (i.e., as regular as possible). This is a trade-off between geometric complexity and physical stability (the more regular a tetrahedron is, the lower energy it has).

Algorithm 11 `PerformCellDivide(c,d)`

```
if $c_{surface} = true$ then
  Find $n \in N(c)$ where $n_{surface} = true$ and $\frac{n_x - c_x}{|n_x - c_x|} \cdot d = \max_{n' \in N(c)}(\frac{n'_x - c_x}{|n'_x - c_x|} \cdot d)$
  if The angle between the vectors $d$ and $n_x - c_x$ is less than $\alpha$ then
    Subdivide the surface edge connecting $c$ and $n$
  else
    Subdivide the surface face the lies in direction $d$
  end if
else
  Find $n \in N(c)$ where $\frac{n_x - c_x}{|n_x - c_x|} \cdot d = \max_{n' \in N(c)}(\frac{n'_x - c_x}{|n'_x - c_x|} \cdot d)$
  if The angle between the vectors $d$ and $n_x - c_x$ is less than $\alpha$ then
    Subdivide the internal edge connecting $c$ and $n$
  else
    Perform Algorithm 10 (p125)
  end if
end if
```

Discussion The two algorithms for cell division presented above lie at opposite ends of the spectrum. Simplex subdivision only slightly modifies the existing structure surrounding the dividing cell, whereas the balanced cell division algorithm removes the local structure altogether and generates a new one from scratch. This brute-force re-tetrahedralisation inefficiently discards the existing structural information. A more efficient algorithm would use this information in order to quickly restructure the local topology. Appendix B offers insight into such an algorithm, but due to time constraints this research was not pursued in full.

There is a natural tradeoff in the cell division transformation between efficiency and quality of the algorithm. The quality of a cell division algorithm relates to aspects of the post-division configuration, for example: the topological symmetry of the daughter cells, its physical stability (i.e., the energy of the configuration), and its structural complexity (e.g., number of new tetrahedra). Simplex subdivision is efficient but produces configurations that are topologically asymmetric and physically unstable. Balanced cell division, on the other hand, is computationally more complex but produces configurations of better topological symmetry and physical stability (at the expense of adding more tetrahedra).

In conclusion, performing cell division within a tetrahedral mesh is significantly more complex than in a triangular mesh. Cell division in a tetrahedral mesh can be accomplished in a number of ways., with a natural trade-off existing between algorithm complexity and the quality of the new structure generated. This section compared two alternative techniques and offered suggestions for further research paths. The second transformation in SDS3, structural cell movement, is now considered.

6.5 Cell Movement

The cells of an s-morph move around in space as a result of external forces, the constraints of the simplexes, and due to collisions with surfaces (§6.3). As cells move, the simplexes change shape and are occasionally flattened (Figure 6.14). In order to maintain the tetrahedral mesh structure either the movement of internal cells needs to be restricted or the mesh needs to be dynamically restructured. The experiments in SDS2 highlighted that giving cells more freedom of movement was essential to maintaining a structure that is physically balanced. This section describes the method used to dynamically restructure the tetrahedral mesh of an s-morph as cells move. The method detects when a topological movement has occurred between two points of time, rewinds the system to the point of movement, and locally reconfigures the mesh.

Edge-flip in 3D In SDS2 the cell movement transformation is modelled with a simple edge-flip operation. In the operation a cell, a, crosses over an edge, e, which is flipped, resulting in cell a now being connected to cell b (Figure 6.13a). This operation is effective because it is confined to the quadrilateral hull containing e, and thus from the perspective of the local neighbourhood, the hull remains unchanged (Figure 6.13b). The transformation acts locally and structural modifications don't propagate.

When exploring the movement transformation in 3D, a first step is to try to generalise the edge-flip operation from SDS2. Is there such as thing as a "face-flip"? Yes, but unlike the edge-flip operation, a face-flip operation modifies the hull containing the face. This operation is demonstrated in Figure 6.13. Due to structural propagation a face-flip is not an appropriate method for modelling cell movement in 3D[14]. An alternative approach to the cell movement transformation in SDS3 was thus designed and is now presented.

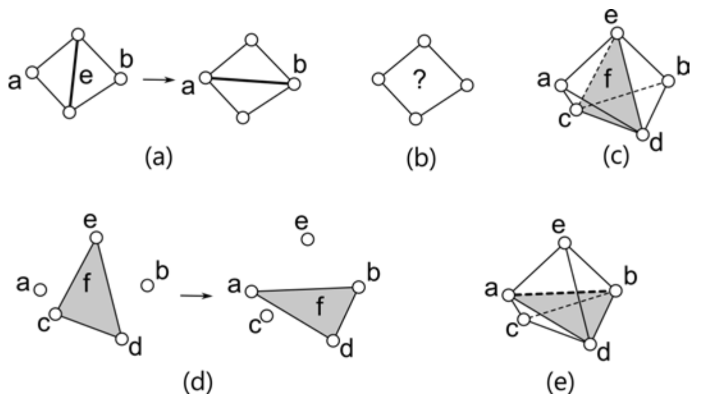


Figure 6.13: Generalising the edge-flip. (a) The edge-flip operation in SDS2 allows a cell a to join to b by flipping edge e within the quadrilateral hull. (b) This operation doesn't affect any cells outside the hull and from their perspective the hull has an unknown structure. (c) In SDS3, the analogous situation to (a) consists of two cells, a, and b, separated by a face, f. The smallest hull containing these elements is the *dipyramid* shown. (d) Consider "flipping" face f such that a and b are now connected, there are a number of ways, one of which is shown. (e) The hull containing a, b and f now has a new edge ab. Comparing this to (c) the hull has changed, and consequently any tetrahedra external to the hull that were connected to edge ec now have to be modified. This may result in further modifications to the tetrahedra connected to them and so on.

Modelling cell movement in SDS3 In SDS3, a topological cell movement is necessary when a cell moves through an opposite face, equivalently, at the instant when a tetrahedron becomes flattened. Therefore, to detect a cell movement, it is sufficient to check, at every time step, whether the signed volume of each tetrahedron (Equation 6.10) has become negative. An *inverted tetrahedron* is one where $vol(t) \leq 0$, and the time of inversion occurs at the instant when $vol(t) = 0$.

Multiple tetrahedra may invert during a single time step. Instead of handling multiple movements simultaneously, only the first movement is performed. This is achieved by rewinding the simulation back to the time of the first inversion, performing the movement transformation, and then restarting the simulation from that point. Given that the simulation is at time step u_2, and the last time step was u_1, for each inverted tetrahedron t, the time u the inversion occurs is computed by assuming linear motion of the vertices and solving for t in $vol(t) = 0$. This equates to solving the following cubic (discarding the roots that fall outside the interval

[14]Note that if the angle formed by the faces connected to edge ab is less than 180 degrees, then a new tetrahedron $acbe$ can be added to "plug the hole", keeping the hull unchanged. In the case where the angle is greater or equal to 180 degrees, a more sophisticated adjustment is needed. These two cases are considered in the algorithm presented here.

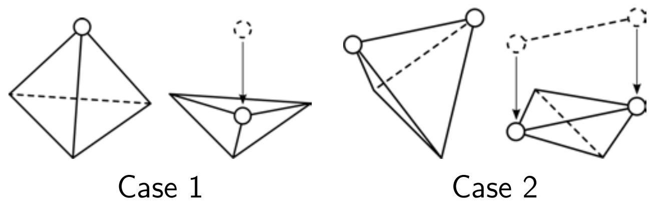


Figure 6.14: Two situations that cause a tetrahedron to become flat. (Case 1) A vertex enters the opposite face of the tetrahedron. (Case 2) An edge intersects the opposite edge. These cases can be distinguished by observing that if any vertex of the tetrahedron lies within the face opposite then Case 1 has occurred, and if not then Case 2 has occurred.

$[u_1, u_2]$):

$$(v_1(u) - v_0(u)) \cdot ((v_2(u) - v_0(u)) \times (v_3(u) - v_0(u))) = 0,$$

where:

$$v_i(u) = v_i(u_1) + \frac{u - u_1}{u_2 - u_1}(v_i(u_2) - v_i(u_1)).$$

Restarting the simulation from the first inversion is not efficient if there are many inversions occurring in one time step. This problem has also been identified in the collision literature and some systems solve it by handling multiple collisions at a time (Bridson et al., 2005). In SDS, however, handling multiple inversions simultaneously is non-trivial because of the topological modifications involved. More specifically, the movement transformations given below assumes that there are no other inverted tetrahedra besides the target one in order to enumerate the topological configurations that arise. The transformations of the two inversion cases illustrated in Figure 6.14 will now be discussed.

Case 1 This occurs when a vertex in a tetrahedron attempts to pass through the opposite face, and results in the transformation described in Figure 6.15. The figure illustrates the case where neither v nor f lie on the surface. If v lies on the surface then the transformation is exactly the same. If f is on the surface then t' doesn't exist and hence new tetrahedra aren't generated. If v and f are both on the surface then the transformation cannot be applied (as this would result in an infinitely thin section). This situation has yet to arise in the simulations performed; however, if it arises in the future then another transformation could be designed.

Case 2 This case occurs when opposite edges in a tetrahedron intersect. Let these edges be named e_u and e_l respectively. Consider the case where e_u and e_l are not on the surface, resulting in a closed hull of all tetrahedra attached to either edge (Figure 6.16). The mesh is transformed at the time of inversion by removing all

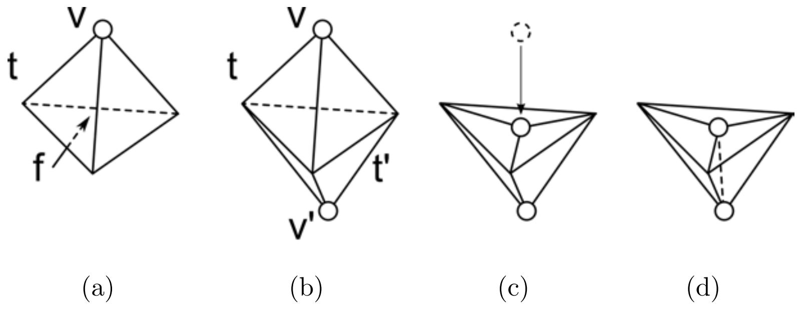


Figure 6.15: Case 1. (a) Consider a tetrahedron t and a vertex v. Let f be the opposite face. (b) Let t' be the tetrahedron that is joined to face f, and v' be the vertex in t' that is opposite to f. (c) v is assumed to have just intersected face f. (d) We remove f and t but keep the faces adjacent to v. We then add a new edge connecting v and v' thus implicitly tetrahedralising the hull of $t \cup t'$.

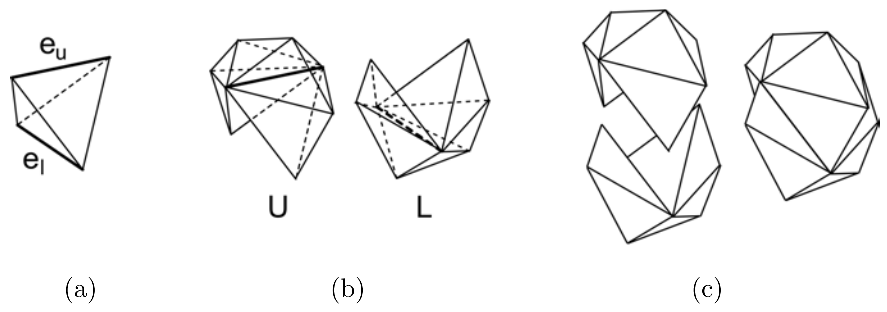


Figure 6.16: Case 2. (a) Consider the tetrahedron in the diagram. Call its upper edge e_u and its lower edge e_l. (b) Create the sets U and L by considering all tetrahedra that share edge e_u and e_l respectively. (c) Consider the hull around the union of those sets, $U \cup L$. When the tetrahedron's volume becomes zero the task is to tetrahedralise the hull of $U \cup L$.

tetrahedra within the hull of $U \cup L$ and tetrahedralising the empty space formed. Tetgen was used to generate the Delaunay tetrahedralisation of the empty hull. In general any tetrahedralisation approach will work, with a tradeoff existing between running speed and quality of tetrahedra. If e_u or e_l are on the surface then the hull is not closed, and additional tetrahedra must be added to cover each edge before applying the method above.

In conclusion, cell movement in SDS3 is considerably more involved than in SDS2. Algorithms were proposed that allow a tetrahedral mesh to change its structure as vertices are pushed around inside. The adaptive structure of an s-morph allows stresses to distribute throughout the mesh and gives cells more freedom in their movement. The algorithm proposed for Case 2 is, however, not very efficient. The hull shown in Figure 6.16c has a very specific structure that is currently ignored by using a general tetrahedralisation algorithm. Efficiency gains could be made by taking advantage of this specific structure and designing a method for tetrahedralising it. Avenues for further improvement to this adaptive aspect of SDS are discussed in §9.2.3.

6.6 Morphogen Model

SDS3 implements a similar morphogen model to the one used in SDS2 (§4.6). The growth program specifies a set of m morphogens, $\Phi = \{\phi_1, \ldots, \phi_m\}$, each with corresponding diffusion and decay coefficients (D_{ϕ_i} and C_{ϕ_i}). Each cell contains a continuous amount of each morphogen, $c_{\phi_i} \in [0, \mathrm{vol}(c)]$. In 3D the morphogen equation is discretised over the tetrahedral mesh in a similar manner to the 2D model (§4.6). The morphogen update equation for a cell c with a morphogen ϕ is:

$$c_\phi(t+h) = c_\phi(t) + \frac{\partial c_\phi}{\partial t} h \text{ (clamped to the range } [0, \mathrm{vol}(c)]) \tag{6.25}$$

$$\frac{\partial c_\phi}{\partial t} = D_\phi \nabla^2 c_\phi - C_\phi c_\phi \tag{6.26}$$

$$\nabla^2 c_\phi = \sum_{n \in N(c)} \frac{\min(n_\phi, \mathrm{vol}(c)) - \min(c_\phi, \mathrm{vol}(n))}{|c_x - n_x|} \tag{6.27}$$

In Equation 6.27 the use of the *min* function in the numerator limits the amount of morphogen that travels between two cells of different sizes. As an example, consider two cells, a and b, with $\mathrm{vol}(a) = 1$, $\mathrm{vol}(b) = 2$, $a_\phi = 1.0$ and $b_\phi = 1.2$. As a is at capacity and $b_\phi < a_\phi$ no morphogen should flow between the cells. This is supported by the model, as $\min(b_\phi, \mathrm{vol}(a)) - \min(a_\phi, \mathrm{vol}(b)) = 1 - 1 = 0$ and hence $\nabla^2 c_\phi = 0$.

6.7 Summary

This chapter presented designs for each of the SDS components in 3D. The physical model extends the SDS2 model by adding tetrahedral forces that preserve local volume, resulting in more solid and stable forms. A number of technical considerations regarding the physical simulation were also discussed, including a simulation rewinding scheme, the incorporation of an adaptive step-size, and the conversion of a surface mesh to an s-morph. Perhaps the most challenging aspect of implementing SDS in 3D is modelling cell division in a tetrahedral mesh. This is due to both the complexity of the tetrahedral mesh structure and the difficulty in visualising (mentally and technically) structural changes in a tetrahedral mesh. This chapter proposed a number of methods for modelling cell division, and discussed properties of these methods, such as quality and efficiency. An algorithm for cell movement was also proposed, illustrating how a tetrahedral mesh can dynamically adapt its structure to accommodate the movement of cells. Lastly, it was shown how to discretise the particle diffusion equation on a tetrahedral mesh, supporting the implementation of morphogen transport. Having examined the machinery of SDS3, the next chapter presents examples of using the system to generate 3D forms.

Chapter 7

Experiments in 3D

Using the 2D limb growth experiments (§5.1) as a basis, the next goal was to generalise them into SDS3. The 2D experiments were heavily focused on biological modelling; the 3D system, on the other hand, was meant as a creative modelling and animation system, and therefore the growth models were stripped back in order to focus on flexibility and ease of use. The primary growth model demonstrated in this chapter, the *basic limb model*, is a simplified version of the 2D limb bud model. Using this model as a base, it was possible to generate a set of different tentacled forms. In this chapter each of these form creation experiments will be discussed, highlighting features of SDS, such as the re-use of modules, variation, and environmental interaction. The basic limb model is discussed first.

7.1 The Basic Limb Model

The basic limb model forms the basis for the experiments presented here. The initial attempt at implementing the limb model in 3D was unsuccessful, but it is useful to consider why. Figure 7.1 illustrates the simulation output of one of the first experiments performed with SDS3 (also see animation SDS3/3). At this stage in the research it was not clear whether the principles seen in the 2D prototype could be successfully applied in 3D, and as this example demonstrates, the early results were not promising. The difficulty in implementing the model arose because of the simultaneous exploration of division algorithms, growth model parameters, and physical parameters, all while fixing bugs with the simulator. This early failure led primarily to a reconsideration of how cell division was performed in SDS3, culminating in the balanced cell division algorithm discussed in the previous chapter. After these early

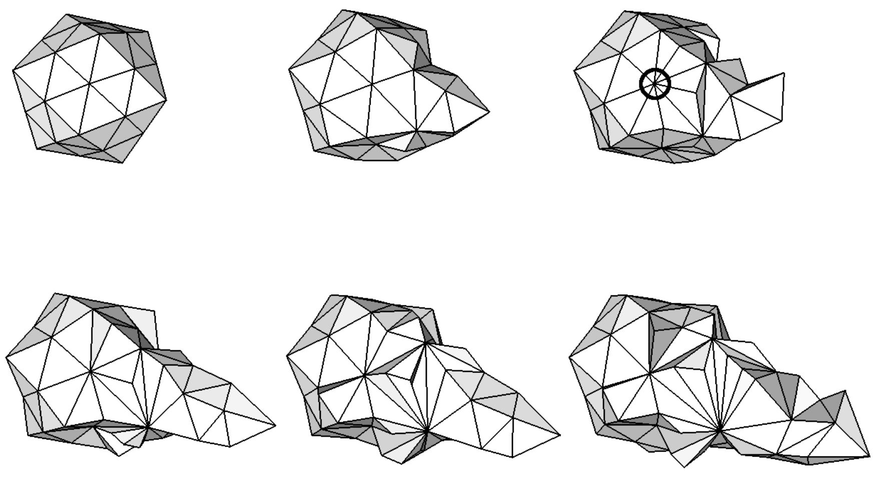


Figure 7.1: (left to right, top to bottom) A simulation sequence of an early limb bud experiment. We can see that amongst other problems, one vertex is far too connected (circled), and the limb lacks symmetry. This failure is due to sing a non-balanced division algorithm and supplying wrong parameters to the growth and physical models.

failures, the limb bud model was stripped back considerably, the balanced division algorithm was implemented, and the initial s-morph configuration was simplified.

The key idea behind simplifying the limb bud model is the elimination of the feedback mechanism. The feedback mechanism is postulated to occur within the biological system, however, as SDS is not modelling biology, this mechanism can be replaced with a simpler one. By allowing the growing tip to perpetually produce morphogen the same effect occurs. This observation removed the need for the second morphogen (Fgf10) which reduced the complexity of the model. The simplified limb bud model is described in Table 7.1. There is now only one morphogen, ϕ, and a binary cell variable $c_t = \{0, 1\}$ is used to differentiate the cells on the growing tip (the AER). Rule r_1 maintains a constant amount of morphogen within the growing tip cells. Rule r_2 directs all non growing tip cell to grow at a linear rate, $\triangle$, if a threshold, K, of the morphogen is exceeded. Rule r_3 causes cells to divide upon reaching a given size, R, with the division directed towards the morphogen source. This basic rule-set results in a localised growth towards a growing tip, due to the orchestration of cell actions, physical effects, and geometric transformations. Figure 7.2 demonstrates the basic sequence of one growing limb (also see animation SDS3/4).

Table 7.1: Cell behaviour rules for the 3D limb growth model.

rule	condition	action
r_1:	$c_t = 1$	$c_\phi = \text{vol}(c)$
r_2:	$c_t = 0$ & $c_\phi > K$	$c_{\Delta r} = \triangle$
r_3:	$c_t = 0$ & $c_r > R$	divide($\nabla\phi$)

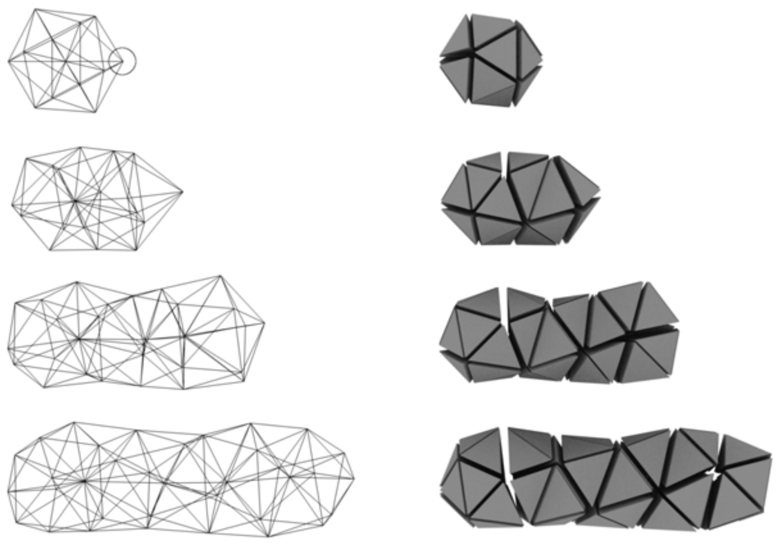


Figure 7.2: A visualisation of how the edges (left) or the tetrahedra (right) of a developing organism change over time. (top) An organism at the start of a simulation has a cell chosen to be a growing tip (circled). From top to bottom: as morphogen diffuses, cells begin to grow and divide near the growing tip, forcing the tip to the right and creating a limb-like structure.

This simplified limb model, while useful, has some limitations. It requires a certain structural configuration in order to develop, specifically, there should be a cell directly below the growing tip (Figure 7.3). This cell divides towards the growing tip and forms the internal *core* of the limb. An analysis of the structural processes behind the growth of a limb in 3D is discussed later (§7.8.2). This point also highlights a major restriction of the current results as the limbs are all just three cells wide. A discussion of this limitation and others is deferred until §7.8.3. The basic limb model shows how coordinating the directed proliferation of a group of cells can generate simple a growth primitive in SDS3. Next we see how limbs can be implanted in existing geometry.

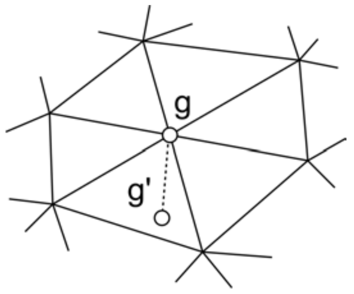


Figure 7.3: A necessary condition for the successful growth of a limb in SDS3, is to have the surface limb tip cell, g, connected to a cell, g', that lies below it under the surface. This growth and division of g' towards g provides necessary internal pressure to push the limb out.

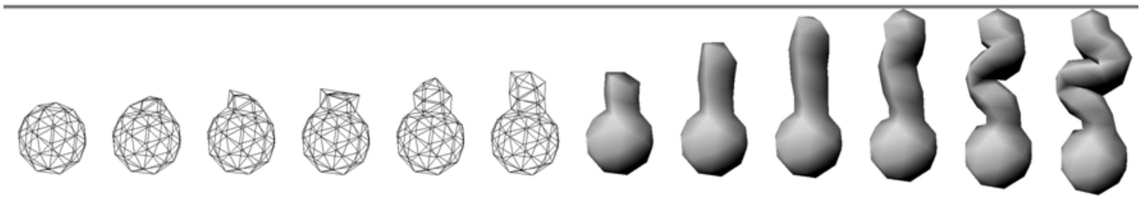


Figure 7.4: (left to right) A sequence from a simulation where a single limb bud was implanted in a sphere. The initial geometric modifications are visible in the wire-frame view. A barrier in the environment prevents the limb from growing through it, causing the limb to buckle and fold in an organic manner, illustrated in the lit surface mesh view.

7.2 Implanting The Limb

The previous section introduced the basic limb model. This section shows how this model can be used, by *implanting* it into an s-morph. Like the limb bud model in SDS2, the SDS3 limb bud model can be incorporated in a design by simply designating a cell as a growing tip. The experiments performed here were set-up using the Blender SDS plug-in (§8.6.3.2), in which a triangulated surface mesh is first constructed. A vertex of this mesh is then designated as a growing tip, and a sub-surface vertex is placed below and connected to the growing tip (as in Figure 7.3). The growing tips are identified using the vertex colouring functionality in Blender, in this example, painting a vertex white indicated a growing tip (as in Figure 7.5). Vertex colours can also be used to assign morphogen quantities or other cell parameters. SDS converts this coloured mesh into an s-morph. Running the simulation results in a single limb growing out of the spherical s-morph, as is shown in Figure 7.4. There are a number of parameters in this model, and the exact values required to achieve specific results will be dependent on the implementation. In practice the parameter values were found experimentally[1] The radius threshold is specified independently for surface (R_E) and sub-surface cells (R_M). A limb can be implanted in any geometric model, or in multiple sites within the same model. The next section demonstrates some forms that have had multiple limbs implanted in them.

[1]The parameters of the simulation shown in Figure 7.4 were $D_\phi = 0.1, C_\phi = 0.05, R_E = 1.1, R_M = 1.2, \triangle = 0.2$, where D and C are the morphogen equation parameters as in Equation 6.26.

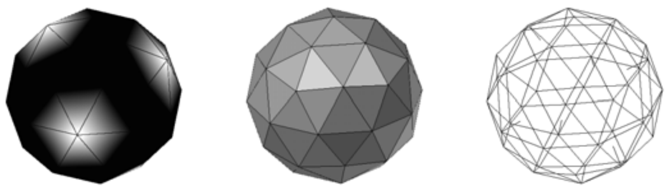


Figure 7.5: Creating a surface mesh in Blender to be converted to an initial s-morph for use in the six-limbed orb experiment shown in Figure 7.6. (left) The vertices of the mesh are coloured to denote type. In this case, white vertices denote limb tips. All the other vertices are painted black, and colour interpolation across the faces show up as a white to black gradient. (middle) The shaded surface mesh, a Blender icosphere (subdivided icosahedron). (right) A wireframe view of the mesh. Examining it closely will reveal the sub-surface vertices that were added and connected to each limb tip as required by the limb model (§7.2).

7.3 Multiple Limbs

The limb model can be re-used by specifying multiple limb tips on a mesh. Using the basic limb program (Table 7.1) multiple limb tips were specified on a mesh (Figure 7.5), resulting in the simulation shown in Figure 7.6 (also see animation SDS3/5). Even though the same limb program is used and the starting conditions are essentially symmetric, it is clear that each limb has a different geometry and, because the growth is constrained within a box, the resulting geometric model is significantly more complex than the initial configuration. Figure 7.7 illustrates another simulation that used the same growth model but differed in the initial configuration, resulting in an octopus-like form (also see animation SDS3/6). Figure 7.8 shows another simulation performed with many more limbs (also see animation SDS3/7).

The ability to re-use components (such as the limb model) is a very useful feature in a creative system as it allows designers to model at a higher-level of abstraction. In SDS, the limb model emerges from low level geometric manipulations, and therefore can be implanted into any geometric model. As the limb develops it literally grows out of the geometric *body*, thus the geometric interface between the two is a complex arrangement of cells. The examples shown so far in this chapter illustrate how, using just the basic limb model, a wide range of forms can be generated with SDS. The next sections presents new growth models which offer variations on the basic limb model.

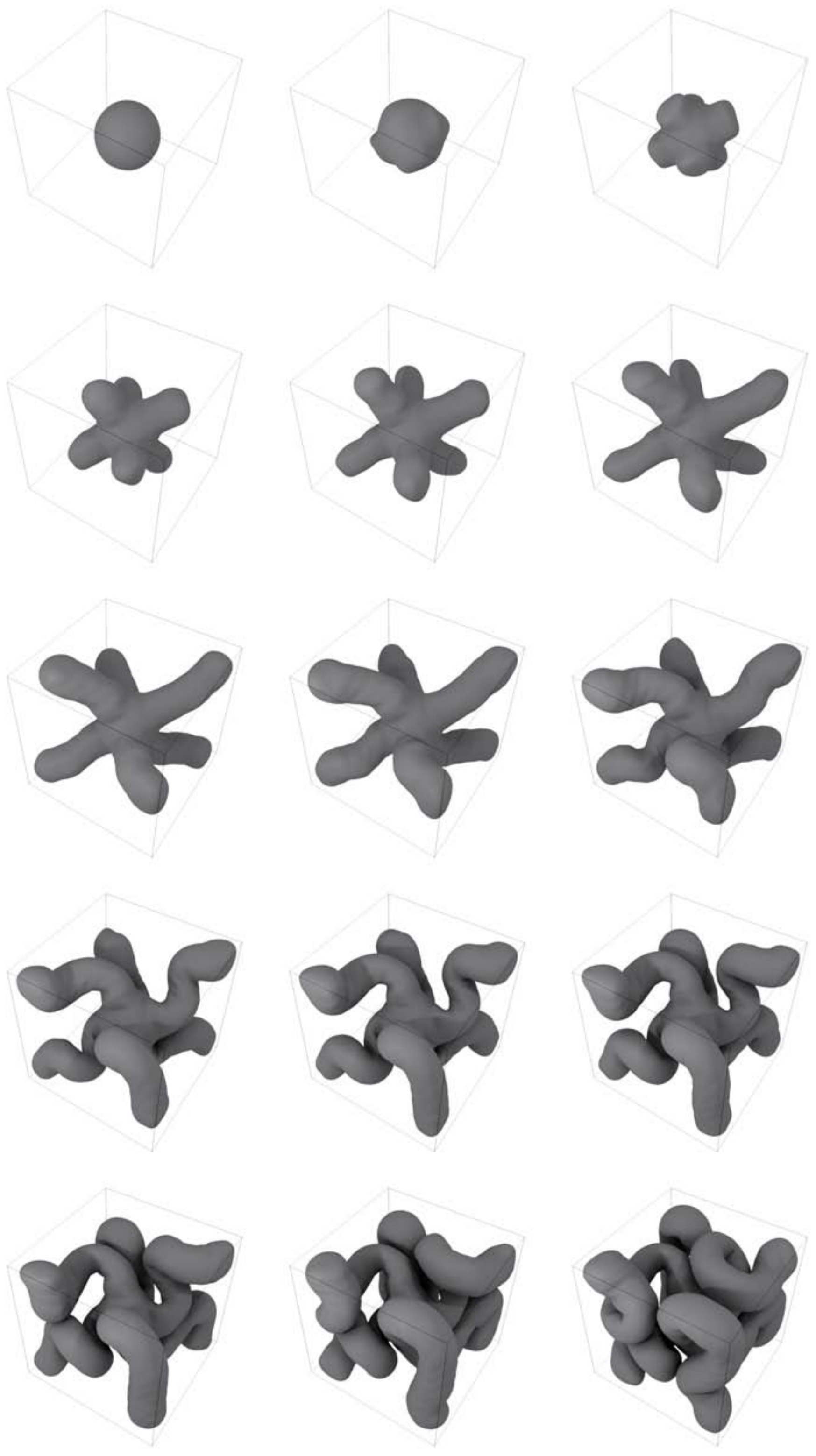


Figure 7.6: (left to right, top to bottom) A developmental sequence of a six-limbed s-morph. As the limbs grow they eventually collide with the bounding box of the environment (shown as a wireframe). They continue to grow in the constrained space and the physical model causes them to curve and bend, eventually filling up the space. Figure A.5 shows a high-res close-up of a similar simulation.

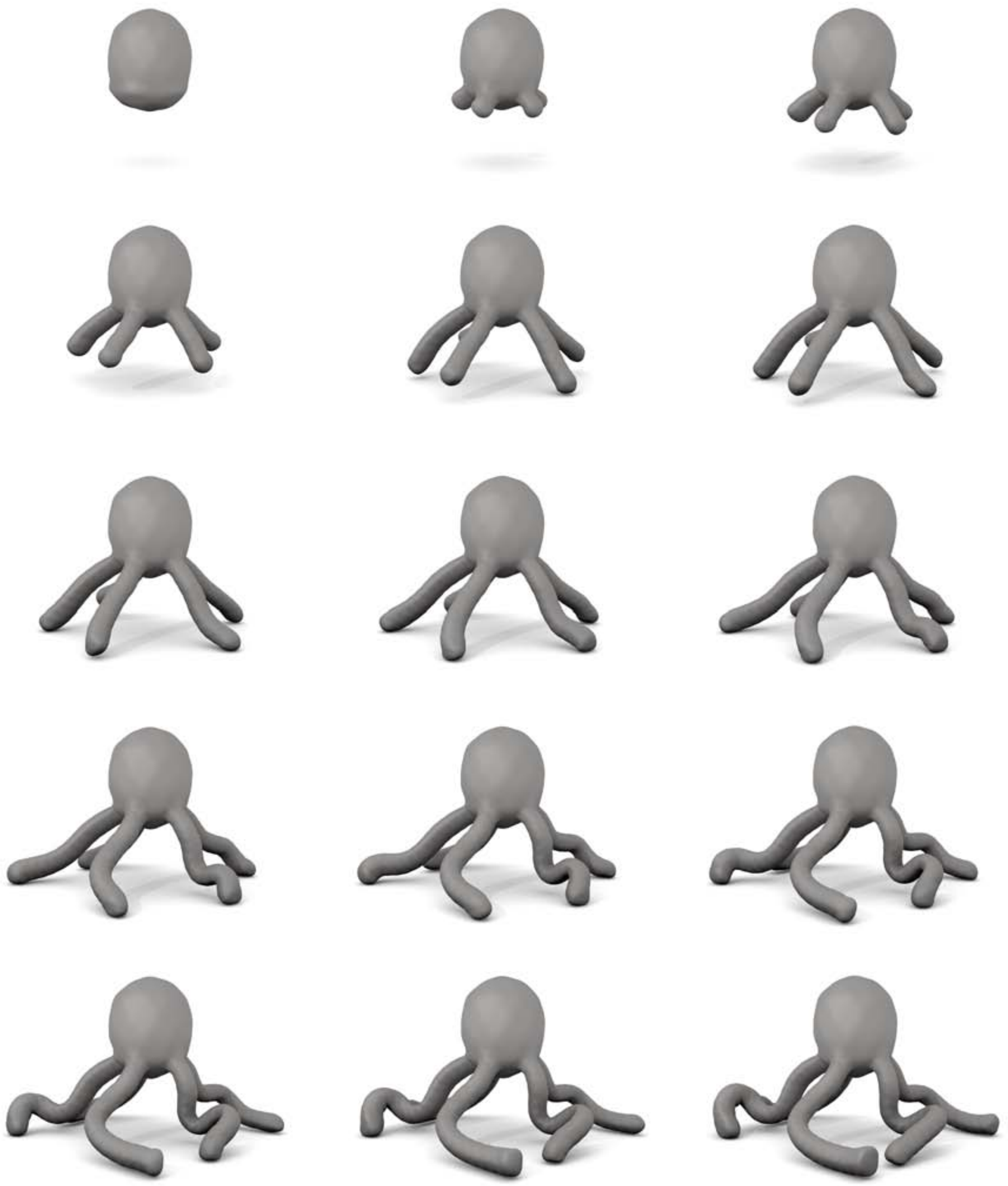


Figure 7.7: (left to right, top to bottom) In this simulation sequence an octopus-like creature (a quinquepus?) is generated by implanting five limb tips into an egg shaped s-morph.

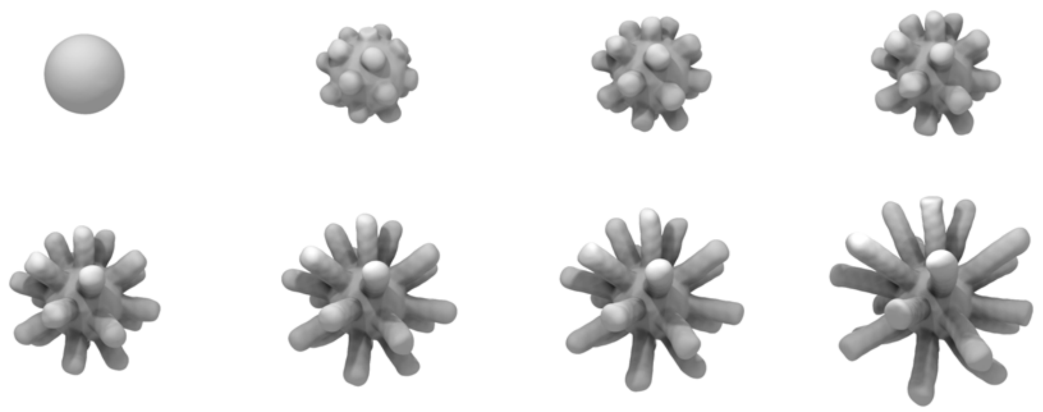


Figure 7.8: (left to right, top to bottom) A simulation of an orb with multiple limb tips implanted around it. The ability to re-use modules, as demonstrated here, is an advantage of the SDS approach.

7.4 Tapered Limbs

Limbs and arms of some organisms (e.g., the starfish) taper along their length. Tapered limbs can be formed in SDS, using the basic limb model, by additionally instructing *all* cells to simultaneously grow at some small rate. This simple addition results in tapered limbs because the threshold of division, R, from the basic limb model, is constant; hence the growing tip remains a constant size while all the other cells grow. This growth model is specified in Table 7.2 and demonstrated in Figure 7.9. The simulation shown begins with a symmetric six-sided form with six growing tips. As the six limbs grow, any non-proliferating cells (i.e., all cells minus the growing tips) grow slowly, causing the body to expand.

One unfortunate side-effect of generating tapered limbs using this method is that the scale of the s-morph increases over time, and therefore it is difficult to predict the final size of the geometric model. While an increase in size is a natural trait of biological development, a dramatic increase in scale may be problematic for a creative system. An alternative method for tapering limbs, that does not suffer this particular problem, is to slowly shrink the end of the limb (instead of growing the body). This approach does not affect the scale of the s-morph but is considerably more complicated. In order to shrink the end of the limb, while maintaining the limb growth process, a number of properties need to change over time, including: the size of growing tip, the radius threshold for division, and the morphogen diffusion and decay rates. Brief experimentation with this model showed that, due to the extra number of time-dependent variables, it is considerably more cumbersome than the original model. Further research into a consistent scale method for generating tapered limbs is thus necessary.

Table 7.2: Growth model for the starfish.

rule	condition	action
r_1:	$c_t = 1$	$c_\phi = \mathrm{vol}(c)$
r_{2a}:	$c_t = 0$ & $c_\phi > K$	$c_{\Delta r} = \triangle$
r_{2b} :	$c_t = 0$ & $c_\phi \leq K$	$c_{\Delta r} = \triangle'$
r_3:	$c_t = 0$ & $c_r > R$	divide($\nabla\phi$)

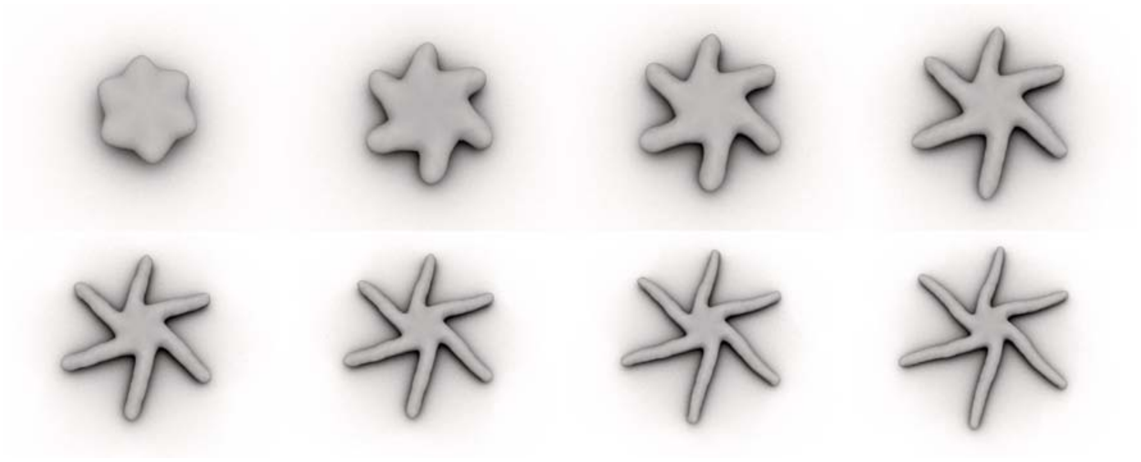


Figure 7.9: (left to right, top to bottom) A 3D version of the SDS2 starfish growth model (Table 7.2). In this simulation, six limb tips are assigned to an initial simple mesh. A rule causes the slow growth of all cells. The radius threshold for proliferating cells remains constant and so tapered starfish arms form. (As the simulation progresses the camera zooms out in order to fit the starfish in frame.)

Once an SDS developmental model is designed it can be applied in different environments. Figure 7.10 shows an early simulation in which the starfish embryo of Figure 7.9 is placed upon a static rock-shaped geometry (also see animation SDS3/10). As the form grows, gravity acts upon it, causing the starfish arms to follow the contours of the rock and ground. In this simulation, the growth rate of the body, $\triangle'$, was too high, causing the limbs to buckle due to friction from the rock. This resulted in an interesting variation of a starfish. Tuning this parameter results in starfish of different shapes (Figure 7.11). Other examples of environmental interaction are explored later (§7.6).

7.5 Stripes and Morphogen Timers

In addition to the geometric output of SDS, texture data can be generated using cell variables or morphogen values in an s-morph. For example, when importing an s-morph into a 3D modelling package, morphogen concentrations in cells can be used to specify vertex colours or texture coordinates. This idea was experimented

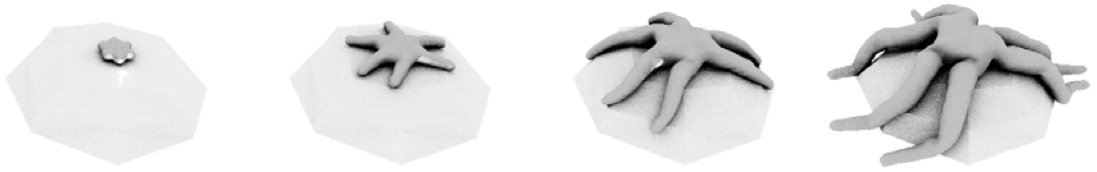


Figure 7.10: (left to right) An early attempt at growing the starfish over a rock. This simulation was initialised with a radially symmetric geometry with six growing tips placed upon a rock. The growth model includes a rule to allow the body to grow slowly over time. This simulation incorporates static friction between the rock and the starfish arms, which results in a buckling of the starfish arms while they grow.

with, in order to generate the striped starfish model shown in Figure 7.11 (also see animation SDS3/10).

The growth model used to generate this striped starfish extended the basic starfish model (used in Figure 7.9) by adding a stripe generating component (Table 7.3). The stripes of the starfish are generated by a *timer* morphogen, ϕ_s, contained within the growing tip. The decaying morphogen in the growing tip acts like a count-down timer. Once the morphogen is depleted the growing tip signals all neighbouring cells to become *stripe* cells, by assigning a *true* value to a custom cell variable c_{stripe}. After stimulating the stripe cells, the timer is reset, and the process restarts. This enables the generation of a number of equally spaced stripes — the distance between the stripes determined by the decay rate of the timer morphogen. In the growth program (Table 7.3), rule r_4 models the generation of stripes. This rule does not use morphogen communication to signal nearby cells to form stripes, but instead allows a cell to explicitly modify the state of its neighbours. This is an example of trading biological realism for modelling flexibility.

Table 7.3: Growth model for the striped starfish model.

rule	condition	action
r_1:	$c_t = 1$	$c_\phi = \text{vol}(c)$
r_{2a}:	$c_t = 0 \ \& \ c_\phi > K$	$c_{\Delta r} = \triangle$
r_{2b} :	$c_t = 0 \ \& \ c_\phi \leq K$	$c_{\Delta r} = \triangle'$
r_3:	$c_t = 0 \ \& \ c_r > R$	divide($\nabla\phi$)
r_4:	$c_t = 1 \ \& \ c_{\phi_s} < \epsilon$	$c_{\phi_s} = \text{vol}(c) \ \& \ \forall n \in N(c) : n_{stripe} = 1$

Morphogen timers can also be used to drive sequences of developmental events. Figure 7.12 illustrates a simulation, derived from the basic limb model, in which a morphogen timer is used to switch between two growth models. In this experiment a timer is added to the growing limb tip. When the timer has fully counted down (decayed) the growing tip is deactivated and destroys all its limb growth morphogen. At this point, the growing tip begins executing a *budding* program, in which it releases a morphogen that causes nearby cells to grow slowly until they reach a

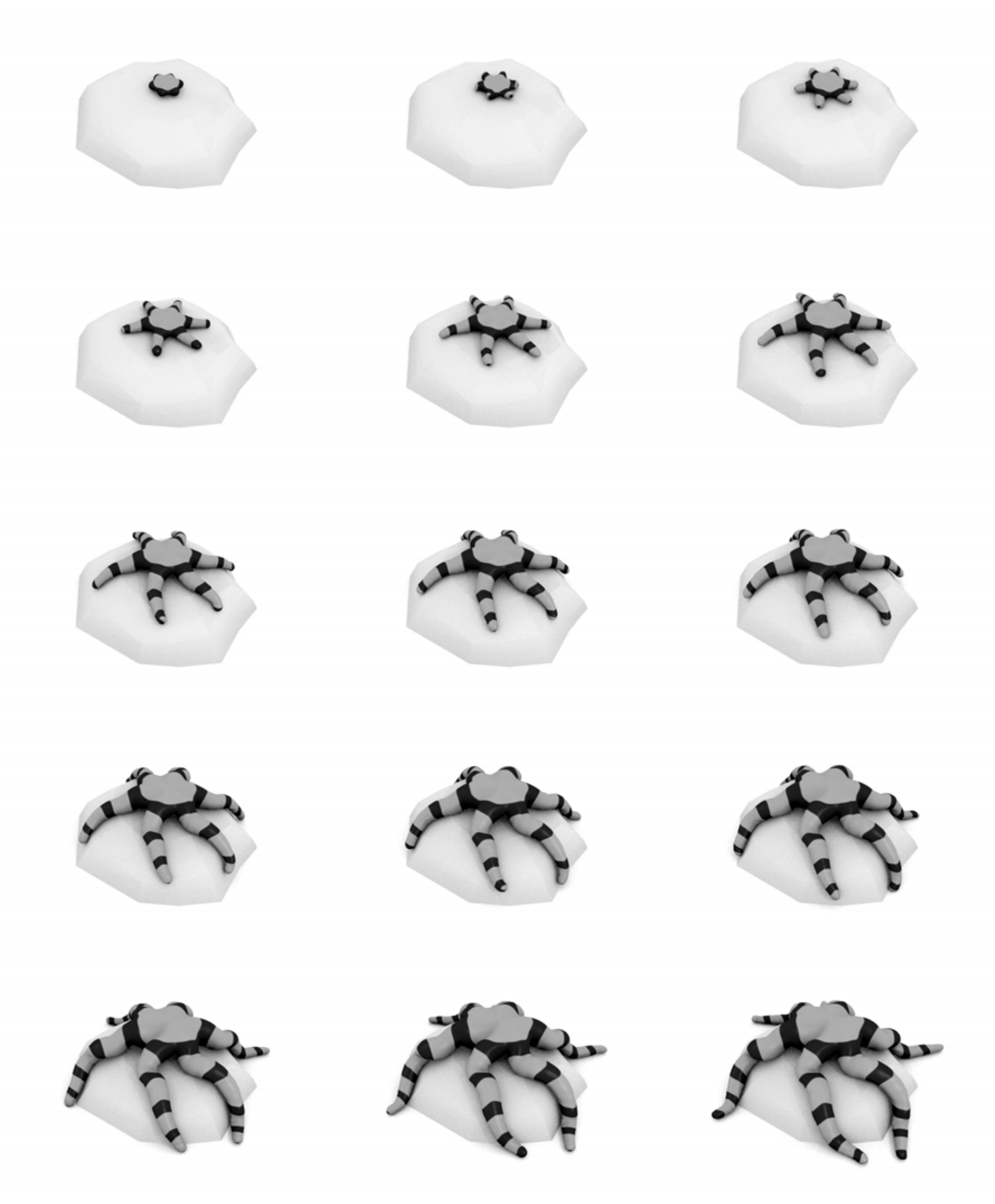


Figure 7.11: (left to right, top to bottom) Simulated growth of a striped starfish on a rock executing the growth model in Table 7.3. This simulation is similar to the one shown in Figure 7.10, with the body growth rate parameter tuned to generate a starfish with more regular arms. An additional rule was added to cause the formation of morphogen stripes.

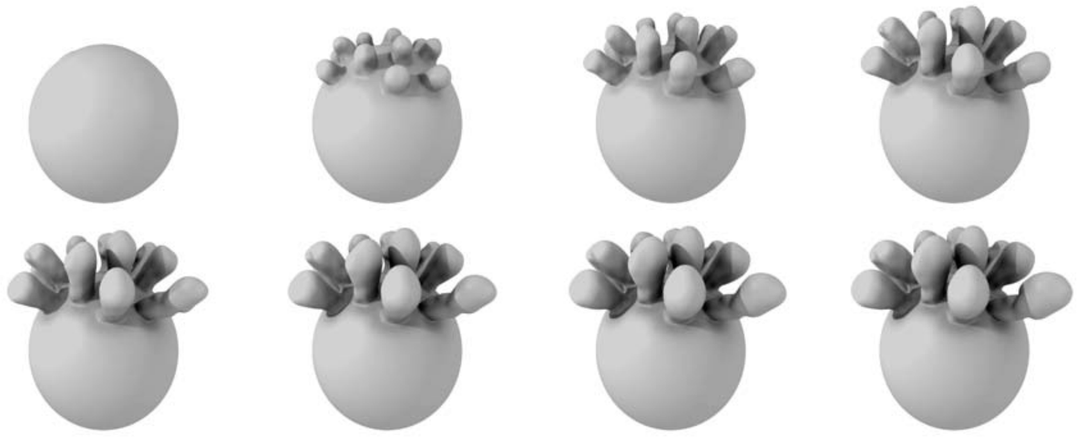


Figure 7.12: (left to right, top to bottom) Simulation of a budding form using the rules in Table 7.4. The growth model consists of a limb growth stage (as seen in the top row), and a bud growth stage (as seen in the bottom row).

specific size. The goal of this growth program was to develop buds at the end of long stalks[2]. Table 7.4 specifies the growth model for the budding limb experiment. The cell type c_t variable can be one of: G growing tip, N normal, T budding tip, B budding cell. B_M is the bud multiplier, B_{GR} is the bud growth rate, and c_{bs} is a cell variable representing the target bud size per cell.

This example demonstrates how sequences of growth models can be modelled. A benefit of modelling cells that have state and can execute logic (as opposed to a continuum of state-less cells (Combaz and Neyret, 2006)) is that complex developmental sequences can be trivially modelled; in the case of SDS, using rule-sets and a model of morphogen transport. Time constraints in this project meant that the potential of SDS to model more complex sequences was not explored. Suggestions for further development in this area are proposed in §9.1.

7.6 Environment

SDS forms are embedded within, and are susceptible to, an environment. Figure 7.13 demonstrates the effect that static geometry has on a growing s-morph. A single limb grows in a constrained environment and will contort to fit within the

[2]An interesting feature used in the budding simulation are the frozen cells that hold the main body in place. The mesh was prepared as usual for a limb bud experiment with growing tip vertices assigned. All the other vertices in the surface mesh were then assigned as frozen (i.e., $c_{frozen} = true$). When the mesh was imported into SDS, the c_{frozen} variable for the internal cells was computed by propagating the variable down into the s-morph using Algorithm 8 (p122). The desired effect of this was to hold the body in place, while the stalks and buds were affected by gravity; however, this is only slightly noticeable in the output due to the short length of the stalks.

Table 7.4: Cell behaviour rules for the budding limb model.

rule	condition	action
Normal Limb Growth		
r_1:	$c_t = G$	$c_\phi = \text{vol}(c)$
r_2:	$c_t = N$ & $c_\phi > K$	$c_{\Delta r} = \triangle$
r_3:	$c_t = N$ & $c_r > R$	divide($\nabla\phi$)
Switch from Limb to Bud		
r_4:	$c_t = G$ & $c_{\phi_2} < \epsilon$	$c_t = T$ & $c_\phi = 0$ & $c_{\phi_3} = \text{vol}(c)$ & $c_{bs} = c_r B_M$
Grow Bud		
r_5:	$c_t = T$ & $c_r > c_{bs}$	$c_{\phi_3} = 0$
r_6:	$c_{\phi_3} > 0.1$ & $c_t \neq B$ & $c_t \neq B$	$c_t = B$ & $c_{bs} = c_r B_M$
r_7:	$c_{\phi_3} > 0.1$ & $c_r < c_{bs}$	$c_{\Delta r} = B_{GR}$
r_b:	$c_{\phi_3} > 0.1$ & $c_r >= c_{bs}$	$c_{\Delta r} = 0$

constraints. This example illustrates that the limb model is not an explicit geometric specification but rather a process, susceptible to external influence.

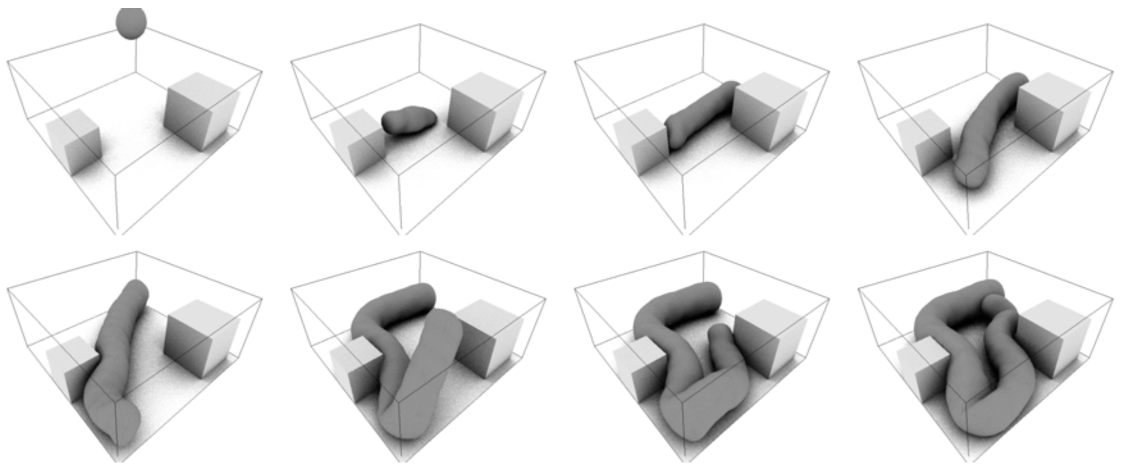


Figure 7.13: (left to right, top to bottom) A simulation sequence of a single limb growing in an environment. The initial conditions place the limb above the ground and it is dropped into the environment. The two blocks constrain the space of the simulation, leading to the development of the form shown. The final result is generated by a combination of a simple growth program, a physical model, and an environment.

Static geometries can be used to constrain the spatial environment. Another environmental factor that can influence the development of an s-morph is an *attractor*. Figure 7.14 presents a simulation within which an attractor is placed into the environment that applies a constant force towards its position, acting on the growing tips (also see animation SDS3/8). This provides a simple control mechanism that guides the directional growth of the limbs. The growth model is specified in Table 7.5. An attraction strength parameter can be used to change the influence of the attractor on the s-morph.

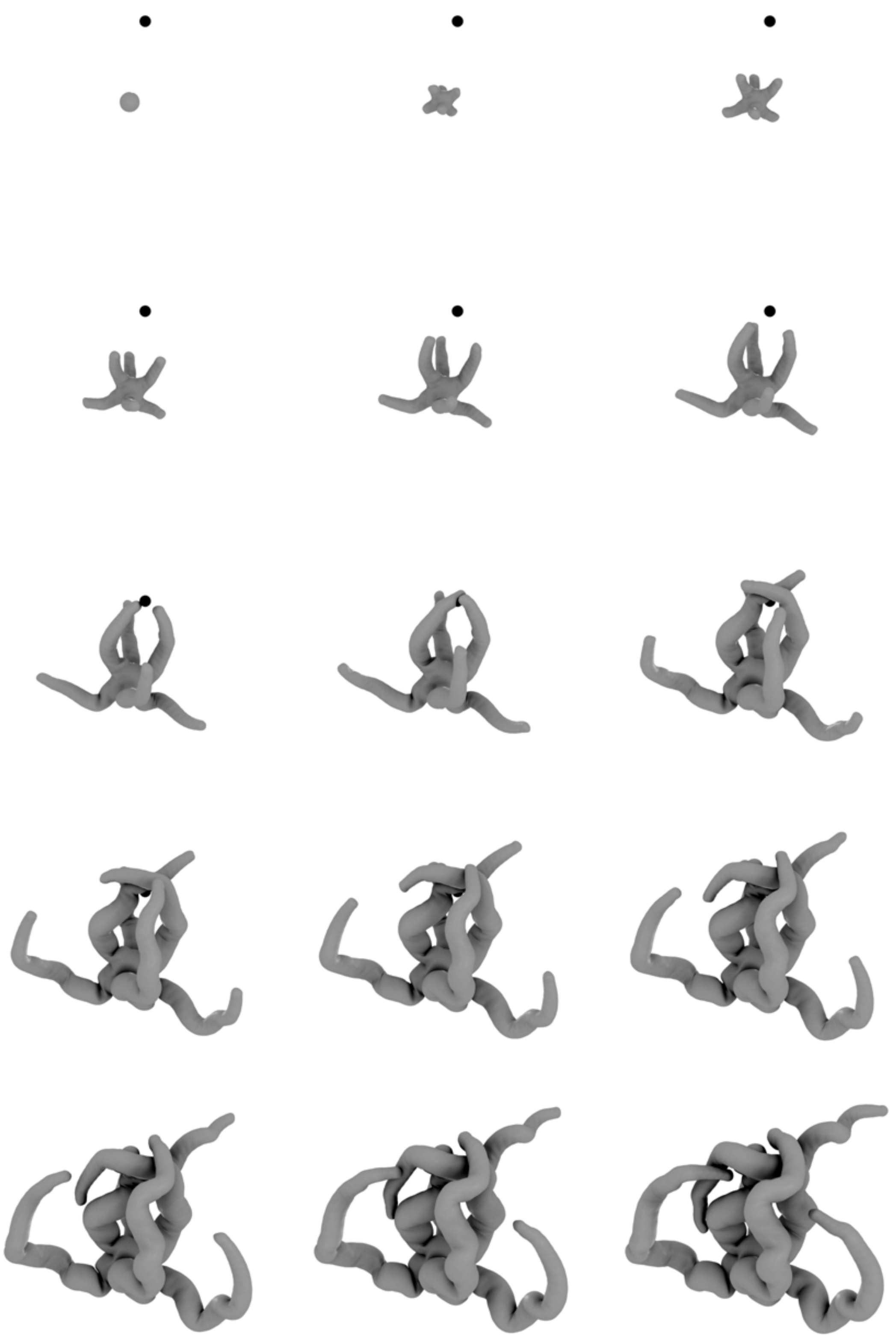


Figure 7.14: (left to right, top to bottom) This simulation incorporates an *attraction point*, shown as a black dot, just above the initial organism. Starting in the same configuration as in Figure 7.6, the growing tips have a small attraction force applied to them, which results in the developmental sequence shown.

Table 7.5: The limb attractor growth model. The attractor is specified with a position, A_x, and strength, A_s. At every time step the attraction force, F_A, is computed for each cell and added to the other forces during the physical simulation step.

rule	condition	action
r_1:	$c_t = 1$	$c_\phi = \text{vol}(c)$
r_{2a}:	$c_t = 0 \ \& \ c_\phi > K$	$c_{\Delta r} = \triangle$
r_{2b} :	$c_t = 0 \ \& \ c_\phi \leq K$	$c_{\Delta r} = \triangle'$
r_3:	$c_t = 0 \ \& \ c_r > R$	$\text{divide}(\nabla\phi)$
r_4:	$c_t = 1$	$F_A(c) = A_s \frac{A_x - c_x}{\lvert A_x - c_x \rvert}$

The SDS framework is flexible enough that many environmental phenomena such as phototropism or chemotaxis could be included within a simulation. A form can be made to *stick* to a geometric object as it grows, allowing vines to be grown on a wall, for example. This embedded physical interaction provides another aspect of user control over the system.

7.7 Curling Limbs

During limb growth in the basic limb model all the proliferating cells divide at roughly the same time and at the same size. This results in straight limbs (unless acted upon by external forces as in the attractor example above.) An interesting *curling limb* can be generated by directing one side of the limb to grow faster than the other. The growth model for a curling limb program is specified in Table 7.6. This model uses the presence of a cell variable, c_{curl}, that affects the growth rate of cells. Figure 7.15 illustrates the growth of a single curling limb. To use the growth program, the growing tip must be set-up as usual, and in addition, the cells neighbouring the tip should be split into two groups with different c_{curl} values. When dividing, the c_{curl} variable is copied to the daughter cells, preserving the local *curliness* in the limb. The reason the growth program is divided into two groups (surface cell rules and non surface cell rules) is merely to improve ease of use. As the limb is one cell thick, it is sufficient to simply shrink the surface cells on one side of the limb. When generating thicker limbs, the c_{curl} variable has to be distributed throughout the limb, and can even be modelled with a diffusing morphogen.

Figure 7.16 shows a more complex curling limb based simulation in which two s-morphs with six growing tips each were placed into an environment together. The growth program then causes them to grow curling limbs. As the limbs collide they wind around each other, creating the complex geometry shown. A close-up of a frame of the simulation is shown in Figure A.9. The surface mesh of the s-morphs from the final frame of this simulation was textured and smoothed to produce Figure A.12,

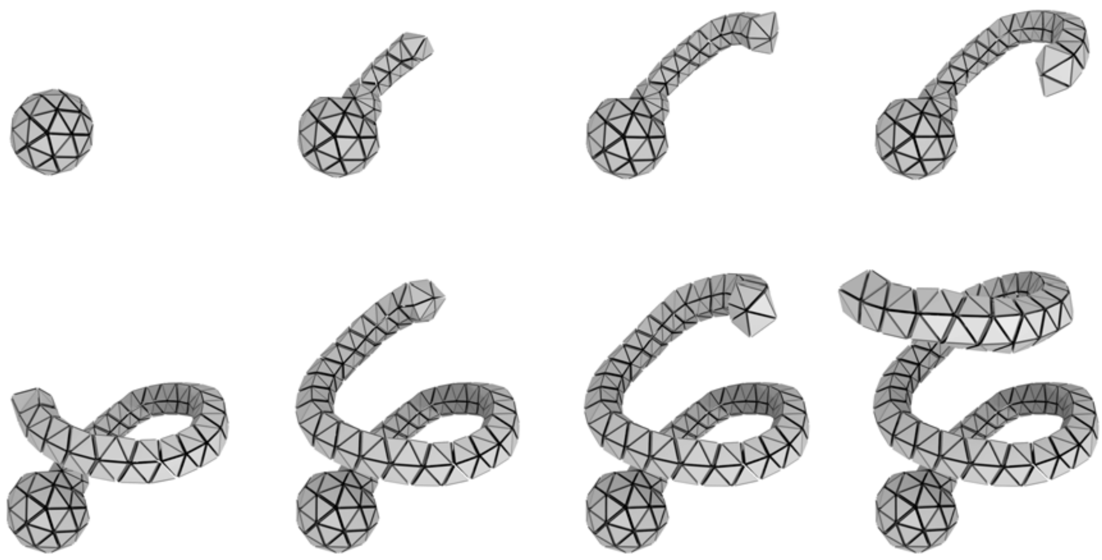


Figure 7.15: (left to right, top to bottom) A curling tentacle can be grown by modifying the limb program. This growth program requires the user to specify the side of the limb to curl around. This was done by painting the cells/vertices on one side of the limb tip a specific colour in the Blender setup environment.

which illustrates how SDS may be used artistically. This example demonstrates that even when using simple growth models, SDS is capable of producing complex, organic, environmentally-sensitive geometry (also see animation SDS3/11).

Table 7.6: A model that generates a curling limb, building upon the basic limb model. A cell variable $c_{curl} \in [0, 1]$ changes the growth rate and division threshold for a cell, allowing different growth rates within the same limb. The amount of curling can be controlled by the curling factor constant F.

rule	condition	action
r_1:	$c_t = 1$	$c_\phi = \text{vol}(c)$
r_{2a}:	$c_{surface}$ & $c_t = 0$ & $c_\phi > K$	$c_{\Delta r} = \triangle(1 - Fc_{curl})$
r_{2b}:	$\neg c_{surface}$ & $c_t = 0$ & $c_\phi > K$	$c_{\Delta r} = \triangle$
r_{3a}:	$c_{surface}$ & $c_t = 0$ & $c_r > R(1 - Fc_{curl})$	divide($\nabla\phi$)
r_{3b}:	$\neg c_{surface}$ & $c_t = 0$ & $c_r > R$	divide($\nabla\phi$)

7.8 Discussion

This chapter concludes with a summary of the benefits of the limb bud model, a geometric analysis of the basic limb model and a discussion of a limitation of the current model.

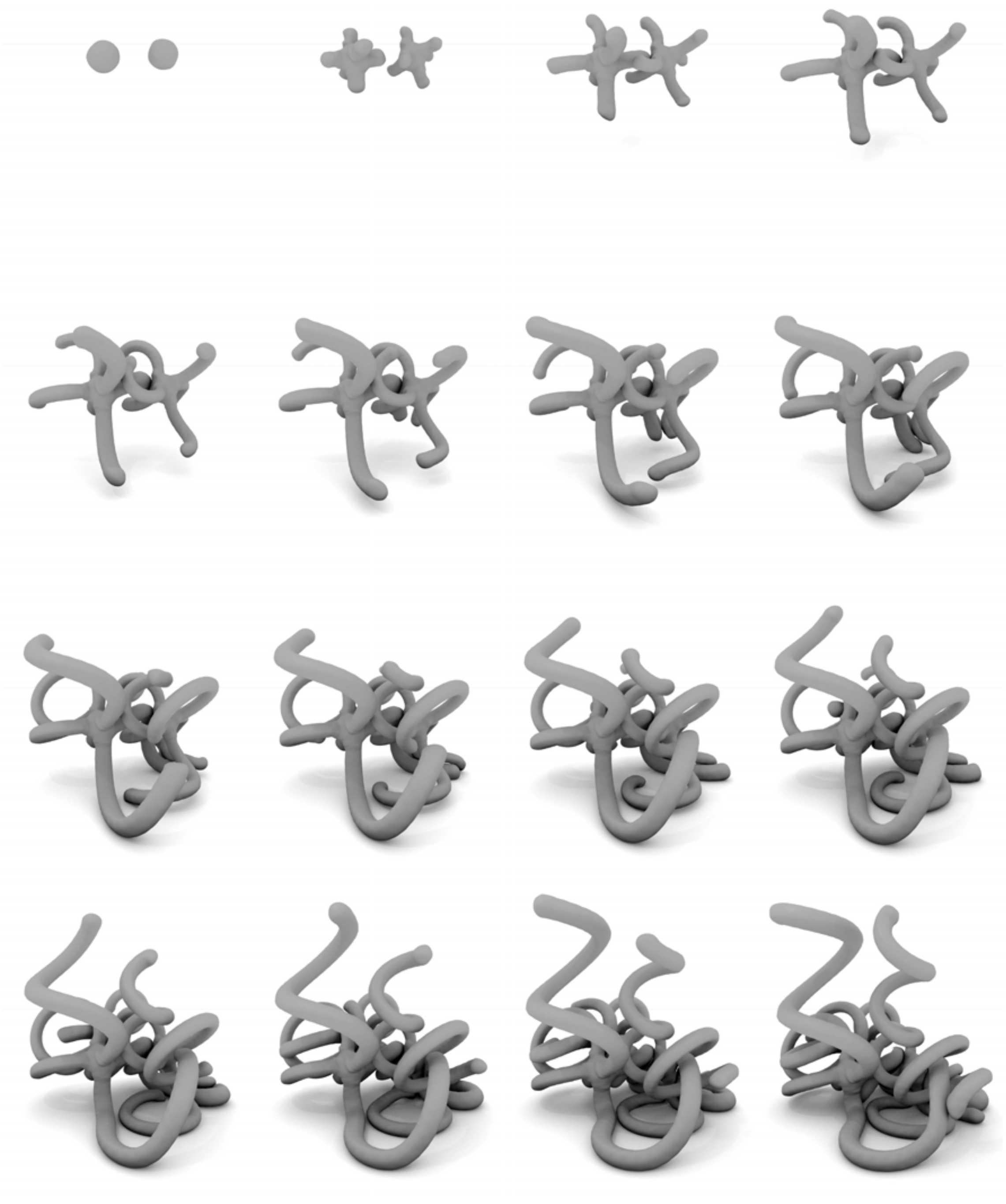


Figure 7.16: (left to right, top to bottom) The simulation of two s-morphs placed within the same environment, both running the curling limb program (Table 7.6). Gravity acts upon the s-morphs as they grow, causing them to fall to the ground. The tentacles of the two s-morphs curl and intertwine around each other, resulting in complex tentacled structures. Also see Figures A.9 and A.12.

7.8.1 Benefits of the Limb Bud Model

A single limb arises from the orchestration of many locally acting processes, including:

- directed cell division,
- cell growth,
- stimulation of the proliferating cells (via morphogen flow),
- cells performing actions based on state, and
- cellular structure changing to accommodate proliferating cells (via physical modelling).

Letting a structure emerge from low-level processes results in a number of beneficial features, including re-use, context-sensitivity, and parametric control.

The limb module can be re-used simply by designating limb tip cells at various locations within a mesh (subject to suitability of local structure (§7.8.3)). This re-use is trivial in a top-down system but is not supported by current embedded generative systems. The process also creates a geometric coupling between limb and body, resulting in an automatically generated natural structural interface. Each of the processes of the limb model are sensitive to local physical structure and environmental conditions. If numerous limb tips are implanted with a mesh, this context-sensitivity means that each limb will be similar, but slightly qualitatively and structurally different. The starfish arms of Figure 5.6 exemplify these characteristics.

Control over the shape of a limb is performed via the parameters of the processes, including physical parameters (e.g., stiffness coefficients), growth model parameters (e.g., morphogen decay rate), and environmental parameters (e.g., viscosity). The parameters of the limb bud model can affect the rate of growth of the limb and its size, and some parameter ranges result in no limbs growing at all or uncontrollable tumour-like growths. This parametric method of interaction is difficult to use because the parameters control the low-level processes and properties of the system, rather than visual aspects of the form. In other words, there is currently no way to directly specify the width of a limb, instead a user is required to "turn the dials" of the growth model until something suitable is created. This indirect level of control is common to process-based systems. The usability of SDS can be improved by designing growth models with fewer parameters that directly control *visual* aspects of the generated forms, for example, a limb model with a *width* parameter would be far easier to use than those proposed in this thesis. Avenues for improving the usability of SDS are discussed in Chapter 9.6.

7.8.2 Analysis of the Limb Bud Model

When considering the high-level model of limb bud formation in biology (§5.1.1) it is easy to see how primitive limbs might form in nature. However at a lower level, when considering the complex arrangements of cells, the processes of cell mitosis, protein signalling, etc., it is not at all clear how these processes orchestrate limb growth. SDS in contrast, is a much simpler system, and it is enlightening to explore how a limb is actually forming at the geometric level. Figure 7.17 illustrates the events that occur during one iteration of extrusion of a limb.

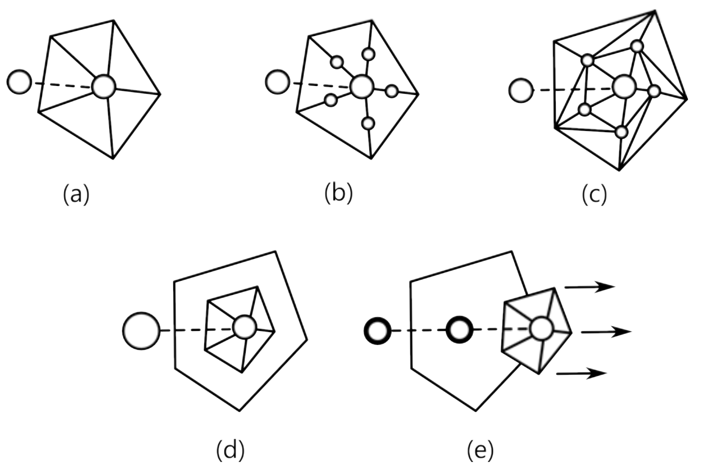


Figure 7.17: The geometric steps involved in one level of limb extrusion. (a) Consider a growing tip, its surface neighbours and the sub-surface cell below it. As the tip diffuses morphogen, the neighbours begin to grow, increasing the lengths of all adjacent edges, including those connected to the growing tip. (b) When the surface neighbours reach the critical radius threshold they divide towards the tip. The division direction is parallel to the edge that connects the cell to the tip, and thus each edge is subdivided (Algorithm 11 (p127)). This happens more or less simultaneously resulting in (c) a configuration such as this one. Importantly, this configuration has a smaller hexagon inlaid within the larger one. (d) Meanwhile, the sub-surface cell is growing, and will eventually divide. (e) When it divides, the edge connecting it and the limb tip is subdivided resulting in the two daughter cells (bold) configured as shown. This is because the division direction is aligned with the edge (Algorithm 11 (p127)). Due to the growth of the new daughter cell and the growth of the neighbours, the inner hexagon is pushed outward. To grow a full limb, this process is repeated.

7.8.3 A Geometric Limitation of the Limb Bud Model

The limb growth experiments presented here generate limbs that are *thin*, internally only one cell thick (see Figure 7.2). The growth model is capable of generating limbs of arbitrary thickness in 2D simply by increasing the rate of diffusion (Figure 5.9), but this is yet to be explored in SDS3. Generating limbs this thin has an important consequence. As discussed above, the limb structure develops because cells are dividing toward the growing tip, pushing it outwards. When generating limbs that are internally one cell thick, we need to ensure that the initial configuration has a cell directly below the growing tip, for without it there would be no internal pressure for the tip to move outwards. To guarantee this constraint, we establish the initial conditions manually by adding a vertex below all the growing tips. An important goal for future work is to eliminate this manual intervention. This restriction has one important consequence for SDS3. It means that currently it is difficult to generate new functional limb tips automatically without the aid of an intelligent system that ensures the correct local conditions are fulfilled.

This issue arises when specifying growth processes at a high-level without having sufficient knowledge of the low-level geometry and topology needed to accommodate them. We can resolve this problem by either adding more geometric detail or specifying the growth processes at a lower level (requiring more detail). Ideally adding more geometric detail would be done adaptively, heuristically determining, for example, which regions of the organism require more detail in order to express specific growth programs and morphogen patterns.

7.8.4 Summary

This chapter presented a variety of experiments performed with the SDS3 system. The limb bud model used in SDS2 was generalised and simplified as the basic limb model which formed the basis for the experiments. Variants of the limb bud model were presented, including a tapered limb, a curling limb, and a striped limb. These variants illustrated different growth model principles that can be applied in other situations. For example, the striped limb uses the concept of a *morphogen timer*, which can be applied in general to unfold timed sequences of events. This chapter also demonstrated the complex effects that embedding has on a simple growth model in 3D: illustrated, for example, in Figure 7.6 in which a six-limbed form is constrained to, and takes the shape of, a cube in which it is grown. This thesis now concludes with a summary and discussion of the advantages and disadvantages of SDS, and suggests further avenues of research.

Chapter 8

Discussion

The aim of this research was to design a system for the automated generation of a class of forms that can be described in natural language as "organic, smooth, soft, squishy, and modular" (§1.2). This thesis presented a new method, the Simplicial Developmental System (SDS), capable of generating this class of form in both two and three dimensions. A number of key features of SDS helped achieve this aim:

- The representation of a system of autonomous cells with an adaptive volumetric structure (the simplicial complex),
- A physical simulation of soft matter that models the deformations that occur from external forces, ensuring the forms appear "smooth, soft and squishy",
- The embedding of a structure into an environment, allowing spatial interactions and relationships to affect development and sculpt the form, and
- Supporting the emergence of form through morphogen-based coordination of cells, and the spatial interaction of the system with its environment.

There are a number of important issues that arose during this research. The remainder of this chapter elaborates on these issues:

- **Structural Operations:** issues concerning different models of cell division and movement,
- **Embeddedness:** a discussion of the consequences of embedding a developing structure into a spatial environment,
- **Simulation:** issues concerning physical simulation, and simulation in general for 3D modelling,
- **Developmental Metaphor:** why the biological models of growth are useful in a generative system,

- **Local Transformations:** advantages and disadvantages of using generative transformations that only transform small parts of a structure, and
- **Implementation:** some key aspects of the SDS software and discussion of issues to consider when implementing a system like SDS.

8.1 Structural Operations

Shape generation in SDS involves processes that affect the “internal” state of cells, the motion of cells, and the structure of s-morphs. The two structural processes or transformations that were examined in this thesis are cell division and cell movement. A notable exclusion from the transformations examined in this thesis is cell death, which would provide a mechanism for reducing the structural complexity. This and other transformations could be pursued in further research (§9.2).

The goal of the cell division transformation is to replace one mother cell in an s-morph with two or more daughter cells, thus adding geometric elements to an s-morph, which are shaped by further processes into cohesive structures. Ignoring cell contents and size (which are distributed evenly amongst daughter cells), cell division reduces to a computational geometry problem: replace one mother vertex with two or more daughter vertices in a triangular (or tetrahedral) mesh. “Biological” constraints on the outcome of the algorithm make it non-trivial: the two daughters should be neighbours, they should be aligned along a direction of division, they should have an equal distribution of topology, and so on.

Allowing the structure of an s-morph to adapt to cell movement supports the development of physically stable configurations. If the mesh wasn’t adaptive, the cells would be restricted in their movement and the stresses would increase. By dynamically adapting the mesh, the stresses can be distributed throughout the mesh. This transformation increases the system’s stability and also results in more natural configurations of cells (Figure 8.1).

The cell division transformations explored in this research offer different approaches to modelling cell division on triangular and tetrahedral meshes (§4.4,§6.4). The proposed methods all replace a vertex in a mesh with one or more new vertices, but produce different quality configurations.

The balanced cell division technique used in the SDS2 prototype (Algorithm 4 (p73)) adds stabiliser cells in order to maintain topological symmetry amongst the daughter cells and increase the regularity (and hence physical stability) of the new triangles. In this algorithm, quality is increased at the expense of more cells and geometric

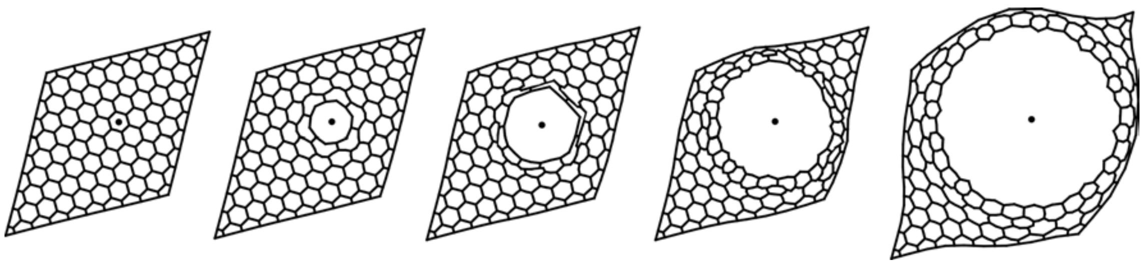


Figure 8.1: (left to right) As a cell grows in an SDS2 simulation, structural cell movements cause the surrounding cells to form an enveloping loop. The number of cells connected to the central cell increases dramatically as it grows.

elements. Simplex subdivision offers an alternative approach that is simpler, produces less geometric elements, but is asymmetric (Algorithm 6 (p78)). The benefit of simplex subdivision is that any k-simplex can be subdivided, and hence the algorithm trivially generalises to SDS3 (Algorithm 9 (p123)). At the very least, this provides a useful starting point and baseline for other cell division algorithms.

The balanced cell division algorithm in SDS2 does not easily generalise to SDS3. As discussed in §4.4.1, the key step to this algorithm is to split the neighbouring simplexes of the dividing cell into two groups and then reconfigure the geometric boundary between them. The structure surrounding a cell in 2D can always be enumerated (it is just a triangular fan). The structure surrounding a cell in 3D however, is much more complex. A 3D version of this algorithm would be more efficient than the brute-force approach (Algorithm 10 (p125)), and would produce better quality results than simplex subdivision. Some research towards this is included in Appendix B.

Although a division algorithm may produce a poor quality configuration, the configuration may not remain poor. If a configuration is physically unstable and has a high energy, the subsequent physics simulation may result in structural transformations in that area. A low quality division algorithm then, can still produce a quality configuration if enough physical simulation time is allowed in order to improve it. Consequently, an SDS simulation can either spend computation time executing a good cell division algorithm, or spend time executing both an inferior cell division algorithm and following this with a period of physical simulation until it improves the configuration. The latter option, however, does not guarantee a quality configuration whereas the former does.

Cell division has been difficult to implement on simplicial complexes primarily because of the way it is modelled: *Cell division adds one new cell.* Adding just a single vertex to a triangular and tetrahedral mesh (such that the other constraints are also satisfied) is a difficult task. Broadly speaking, cell division in SDS exists solely to increase the structural complexity of an s-morph. Considered this way, there are alternative interpretations of cell division, one example is: *cell division subdivides every adjacent simplex with a new cell.* This model ignores the fact that a biological cell divides into two. Compared to the balanced cell division algorithms this model can be implemented efficiently and is perfectly symmetric; however, it results in the generation of a substantial number of new geometric elements, possibly too many to control. An interesting avenue of further research would be to examine more localised structural operations that add new geometric elements. Some operations may be suited to generating specific sub-structures, whereas some may be more general.

8.2 Embeddedness

As demonstrated by SDS and existing systems (Měch and Prusinkiewicz, 1996; Kaandorp and Kübler, 2001; Greene, 1989), embeddedness adds a dimension of complexity to a generative system, at the cost of direct control. Characteristic of embedded and process-based systems is the indirect level of control a user has, and SDS is no exception. These indirect modes of interaction include using static objects in the environment to shape the growth of a form, positioning forces that attract or repel parts of a form, or implementing ad hoc spatial interactions. There are many other ways in which spatial interaction can affect a generative form system. For example, allowing surfaces to fuse together upon impact could allow the generation of complex surface topologies (see §9.2.2). A creeper vine could be modelled by allowing surface cells to *stick* to objects they collide with.

8.3 Simulation

Developmental and physical simulation holds much promise for computer-assisted generation of 3D geometry. In SDS, the developmental simulation is the creative force. As cells grow and split, they add structure and geometry to an s-morph, in a harmony of cellular proliferation. The physical simulation plays the counterpoint to this harmony as a constraining force. It grabs on to cells and slows them down, smoothes out the surface of an s-morph, and glues together the (otherwise free) cells. The balance of these two forces results in forms that have an organic character.

The physical model in SDS is simple and effective, however it can be made simpler by changing from a Newtonian to an Aristotelian physical model. The model proposed in this thesis (§6.3) implements Newtonian physics, in which the motion of cells in an s-morph are governed by the familiar law $F = ma$. Under this model, cells have an acceleration, velocity and position, which means for example, that we can simulate an s-morph blob that is released at some height, falls down at an ever increasing velocity, and then collides with and bounces off the floor. This is a realistic *dynamic* behaviour. A simpler physical model, Aristotelian physics, relates force directly to velocity, using the law $F = mv$. In the Aristotelian universe, acceleration does not exist, and the dynamics of objects are very different to what we observe in reality — masses would move as if in a very viscous fluid. From a simulation perspective however, the Aristotelian model is simpler and more stable, and will still result in organic cellular configurations and smooth forms. If the desired output of SDS is not an animation, but rather a single 3D model at a snapshot in time, then Aristotelian physics is a viable alternative to the Newtonian model. Typically the output required from a generative 3D modelling system is not a realistic animation but rather a single 3D model, considered the "output" of the system, and so exchanging realistic dynamics for simplicity and stability is a reasonable compromise. Some systems don't consider realistic dynamics, such as the "quasi-static" simulation of the semi-interactive morphogenesis system (Combaz and Neyret, 2006). This system produces a developmental sequence of forms, each one simulated to static equilibrium. Simulating a bouncing ball is impossible under this scheme, but the individual forms generated are extremely complex and realistic.

It is unsurprising that simulation is not typically used to model 3D organic form, given the indirect and counter-intuitive control schemes and the time required to perform a simulation. Advances in both these areas will likely increase the uptake of these methods in the creative industries. The first problem can be addressed by offering additional modes of control and user-friendly interfaces (see §9.6). The second problem will be addressed by developments in hardware and hardware-accelerated simulation algorithms that will decrease the time a user has to wait. Most 3D modelling and animation packages incorporate methods for simulating natural phenomenon such as fluid flow, fracturing objects, and soft-body systems. These systems all require a user to wait as the simulation is performed or rendered, and, likewise, it may eventually become acceptable to wait for a simulation when modelling 3D organic forms.

8.3.1 Timing and Complexity

Besides the loss of direct control, a major disadvantage of simulation and embeddedness in SDS is computational complexity. Calculating the spatial interactions of geometric elements is computationally demanding, and so the total time required for

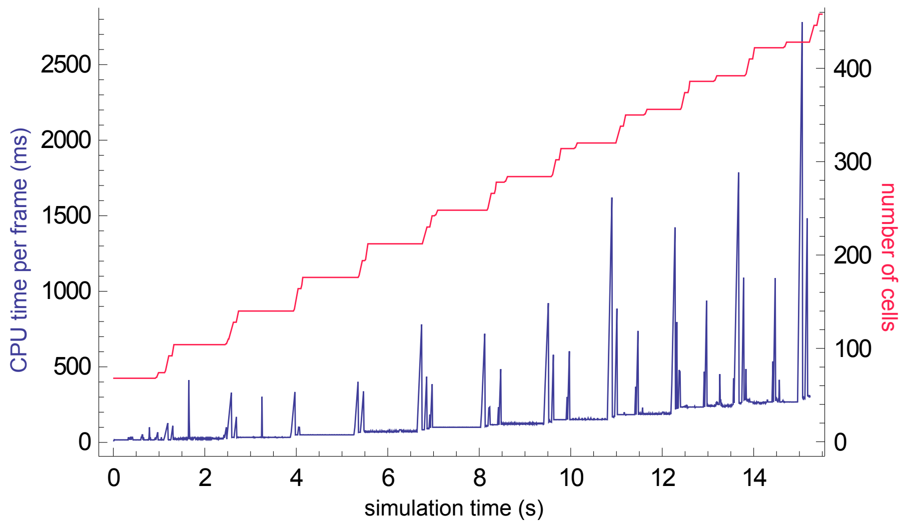


Figure 8.2: Timing information for the experiment shown in Figure 7.11: simulation time on the horizontal axis vs. the CPU time required to compute one time-step (blue) and the number of cells in the s-morph (red). The spikes in the CPU time correspond to cell divisions in the developing s-morph.

simulation is significantly higher than non-embedded systems (such as L-systems). This becomes tedious during experimentation, when parameters need to be modified, and a simulation may need to be re-run many times.[1] The striped starfish (Figure 7.11), for example, took approximately 5.7 minutes to simulate on a standard desktop PC with a dual core 2GHz processor and 2GB RAM. Figure 8.2 shows some timing information for this experiment. The red line shows that the number of cells increases linearly over time and cell division events for each limb occur more or less simultaneously (due to the symmetric initial conditions). The blue CPU time plot has two main components: the base-line time to simulate a frame (which includes calculations for physics, morphogens, collisions, etc.), and large spikes in computation time which correspond to cell divisions. Although the spikes are quite conspicuous, cell divisions are relatively infrequent (in this experiment at least), and so it is the base-line that contributes the most to the total simulation time.

While a simulation in SDS is computationally expensive, it can be argued that the visual complexity and realism outweighs this cost. As hardware improves this cost will decrease, and generative systems that operate in fully simulated physical environments will become an important aspect of 3D simulation-based geometric modelling.

[1]Most of the experiments performed in this research were simulated overnight. A batch script was written to sweep over a range of parameters, thus running the experiments sequentially. The results were then analysed the next day.

8.4 The Developmental Metaphor

The processes that drive the generation of a form in SDS are inspired by real processes that occur during biological development. Of the key processes involved in morphogenesis (Gilbert, 2006, p13), SDS models cell division, cell movement, cell growth, and changes in the composition of cells and secreted products. All these processes were crucial in SDS in order to grow limbed forms. The cellular processes not modelled in SDS include cell death and changes in cell shape. A consequence of omitting these processes is that SDS cannot directly adapt some biological models of structure formation (e.g., autopod development (§9.2.2)). There may, however, be alternative developmental pathways to the same result, so this is not such a big disadvantage. Incorporating some of the morphogenetic processes into SDS provides confidence that SDS is capable of reproducing some interesting developmental phenomena. This thesis demonstrates that even with a few processes, some interesting forms can be generated.

While developmental biology inspired this research, it is not based purely in theoretical or computational biology, and so SDS is not obliged to accurately simulate biology. Biology is simply a starting point, upon which useful features can be added to the system. The further the divergence from the developmental metaphor, the less reliance can be placed on the observation of biological processes to help build generative models. One useful feature, for example, is that SDS cells can access their absolute world coordinates — this has no direct analogue in biology, nor physics.[2] This feature could be used in many ways, for example in a model of tree growth, to allow longer branches to grow the lower they are to the ground. This kind of feature may increase the expressiveness and useability of SDS; however, if developmental modelling is to become a competitive technique for building complex geometry, the research focus should be on broad generative mechanisms and how to use them, rather than ad hoc features suited to a small number of cases. These mechanisms may take the form of axiomatic processes, such as those used in the limb bud model in SDS. Suggestions for further development in this area are offered in Section 9.1.

8.5 Local Transformations

One very important property of developmental (and generative) systems is local transformation. Acting locally, a transformation needs a minimal amount of information, as it doesn't need to account for the entire structure. The cumulative

[2]According to the theory of relativity there is no absolute universal coordinate system.

effect of many of these local transformations results in the massive difference in information between the input (the transformation rules) and the output (the entire structure). The structural transformations in SDS act locally, affecting some small region of an s-morph.

Generating large cohesive structures requires a balancing act of coordinated processes. Creating structures with simple hierarchical or fractal relationships is typically easy, as the coordination amongst processes is vertical or hierarchical, i.e., information is passed down from parent to child, from space to sub-space, etc. Context-free L-systems can only model vertical coordination, and yet can still generate many interesting structures. *Horizontal* coordination between sibling cells, or across a structure, lies at the opposite end of the spectrum. This mode of coordination is demonstrated succinctly with cellular automata, in which the only mode of communication is typically between immediately adjacent cells. Longer range coordination in CAs can emerge through morphogen patterning (Turing, 1952), dynamic structures (e.g., gliders in the Game Of Life (Berlekamp et al., 1982)), or physical models (Cickovski et al., 2005), for example. These models are typically expressed as CA rules, and hence act locally. They are all methods by which transformations can be coordinated to form macro-level structures or dynamics.

This thesis demonstrated that SDS is capable of generating macro-level structures from coordinated local processes. This is due to a number of mechanisms which coordinate the cellular actions and structural transformations, including morphogen diffusion through an s-morph, physical modelling of material, and spatial interaction within an environment. Each of these components contributes a different organising force, but all are necessary to achieve the variety of forms shown.

Brittleness In some systems it is possible that effects from a single transformation may propagate throughout a structure. An extreme example of this is the *Game Of Life* CA (Berlekamp et al., 1982), in which changing the state of a single cell can have drastic consequences on the entire system, in other words this system is very *brittle*. In general, the propagation of local effects make it very difficult to build cohesive structures, so the control over this propagation should be considered.

In SDS the immediate effects of division and cell movement are confined to a relatively small area. It was found in SDS2, however, that a single cell movement would occasionally result in a series of cell movements, following the path of least resistance through the s-morph until the boundary was reached. This seemed to occur more often as the system became more stressed, or equivalently, had a higher energy. The propagation was typically confined to a chain of cells and hence was not too disruptive. To state a hypothetical example: imagine that the SDS2 starfish form is under greater stress because the spring coefficients are extremely high, if a limb on one side collides with a wall, a change of cell movements may propagate through the starfish, causing a cell to pop out of a limb on the other side! It is interesting to note that as an s-morph is placed under greater stress, the system becomes more brittle and the propagation of transformation effects increases. This brittleness can be attenuated by increasing the physical stability of transformations and distributing stresses appropriately (using e.g., adaptive meshing (§9.2.3)).

Comparison to high-level modular systems Consider a case in which instead of having modules emerge from local processes, they are specified at a higher level. Such high-level approaches to design are easy to use and can produce a wide-range of natural looking forms, typically with interesting symmetries and hierarchical modular relationships (e.g., Lintermann and Deussen, 1998; Maierhofer, 2002). Specifying modules at a high-level, however, has some disadvantages. Figure 8.3 shows an illustration of such a problem occurring in a high-level rule-based geometric modelling program. In this system, there are two modules, *branch* and *trunk*. The branch can be attached to the trunk by specifying a hierarchical relationship between the two. The problem then arises: *how* is the branch attached to a trunk? In nature, branches typically smoothly emerge out of trunks. To achieve a natural attachment between modules is a difficult problem when each module has a precise high-level specification. A branch module is attached to a trunk module, but the geometric interface between them is not natural at all. The main problem is that the relationship between branch and trunk has been abstracted away where *branch* and *trunk* are arbitrary objects and their relationship is purely spatial. The fact that in natural systems, *branches grow out of trunks* has been abstracted away, and one of the consequences of this abstraction is the lack of a natural geometric interface. This interface problem has been solved specifically for tree models (Bloomenthal, 1985; Holton, 1994), and can also be rectified using ad hoc procedures, for example, geometrically smoothing sharp edges thus reducing the artificiality of the geometry. There are, however, many other developmental and physical phenomenon that cannot be modelled at this level, for which a low level approach like SDS is far more suited.

8.6 Implementation Issues

SDS is a complex system that required the implementation of many different components. The physical simulator, structural transformation algorithms, morphogen

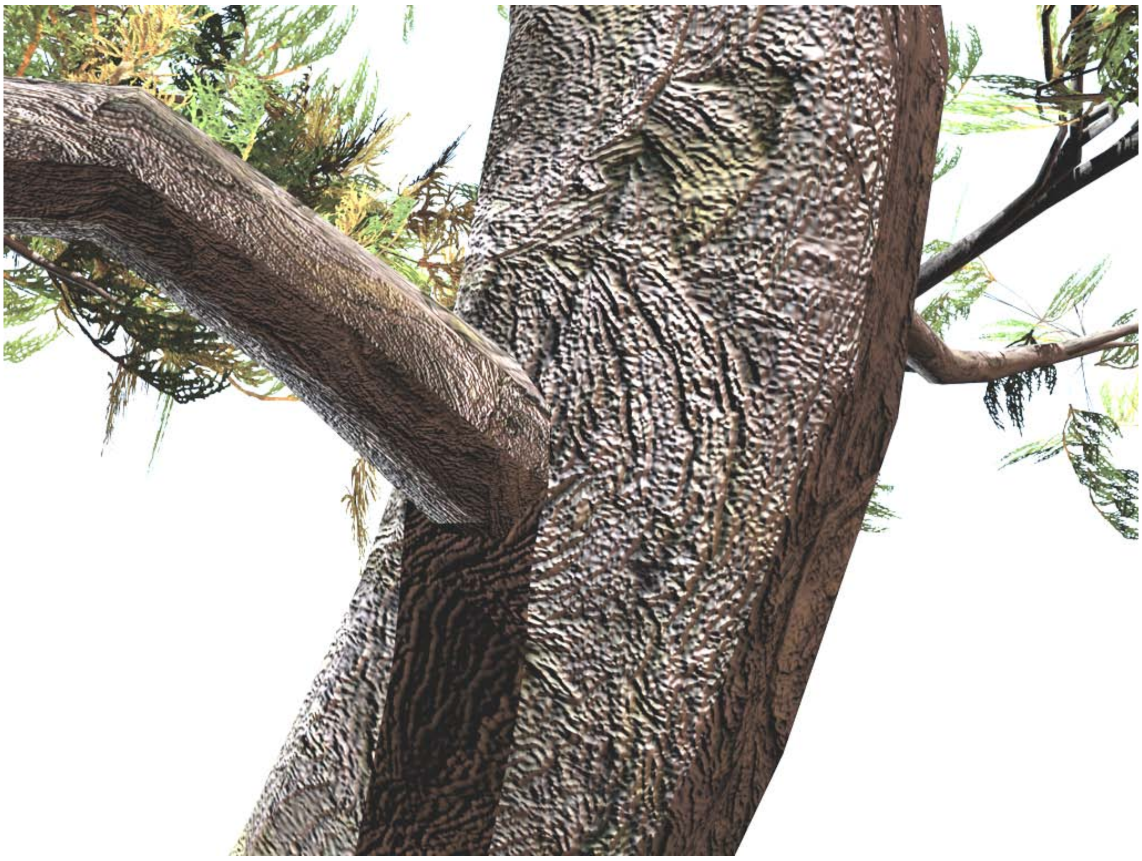


Figure 8.3: A 3D tree model built with XFrog (*Cuspressus* example from `http://www.xfrogdownloads.com/Walli/` retrieved on 25/03/2008). The geometric intersection between the branch and trunk modules is not natural, and is exacerbated by the difference in texture. This occurs because the system models the relationship between the modules as a simple affine transformation, and ignores the real complex developmental history which binds trunks and branches in nature.

simulator, and cell programs, all needed to be robust, efficient, and act harmoniously with each other. A significant portion of this research was spent building and debugging the two SDS simulators and the supporting scripts and tools. The simulation nature of SDS meant that some software bugs only appeared after long periods of simulation, resulting in a lengthy code debugging cycle. This section discusses features of the simulation software, the problems encountered during the software development and the solutions designed to address them. Hopefully, this information will provide insight into the implementation difficulties associated not just with SDS, but other simulation-based developmental systems.

8.6.1 Debugging

Much of the time spent on this research was divided simultaneously between developing software, designing the algorithms within SDS, and defining, simulating, rendering, and analysing the experiments. Developing the software consumed much of the time, since most of the system was built from scratch. Moreover, debugging a simulation system like SDS can be tedious as often a bug might only manifest itself after many minutes, or sometimes hours, of simulation. It was found helpful to run the simulator in a *debug mode*, in which for each frame, useful information was logged, and mesh consistency checks were performed. The logs were used to trace any bugs that appeared and included information regarding cell divisions, mesh conditions before and after structural transformations, collisions detected and handled, and s-morph energy levels. The mesh consistency checks were particularly useful as they ensured that the structural transformations didn't invalidate the mesh data structure.[3] Operating in debug mode slowed the simulations down considerably, resulting in a cycle of designing, running, rendering and analysing an experiment that spanned a number of days, slowing progress considerably, especially as only one simulation could be performed at a time on the single machine used.

Bugs in the structural modification algorithms were especially onerous to analyse in SDS3, due to the difficulty of visualising a dynamic tetrahedral mesh. This can be assisted by restrictively viewing the structure, e.g, just the edges, cells, or neighbours of a selected cell. Additionally, simplex topology graphs can also be helpful (see Appendix B). Nonetheless, a significant difficulty with implementing

[3]The structural algorithms directly manipulate the mesh data structure, for example adding vertices, connecting vertices together with edges, deleting tetrahedra, etc. It is their responsibility to maintain a valid data structure. For example, if a tetrahedron is deleted, then all appropriate edges must also be deleted, and neighbourhoods of cells updated. The mesh consistency checks are a brute-force, debug-mode-only method to ensure mesh validity.

SDS3 was that tetrahedral meshes that change shape and structure are very difficult to both conceptually and technically visualise.

8.6.2 SDS2

The SDS2 simulator was developed during the formative stage of SDS, and was critical in the development of the ideas that led to the general SDS framework and SDS3 system. The SDS2 simulator is interactive and allows users to manipulate an s-morph in real-time. Cells can be manually moved and instructed to divide. This interactivity was very helpful in understanding how division and structural movement transformations affected the surrounding structure of an s-morph. It is particularly interesting to watch the mesh dynamically adapt as a cell is moved around within it (see animation SDS2/2).

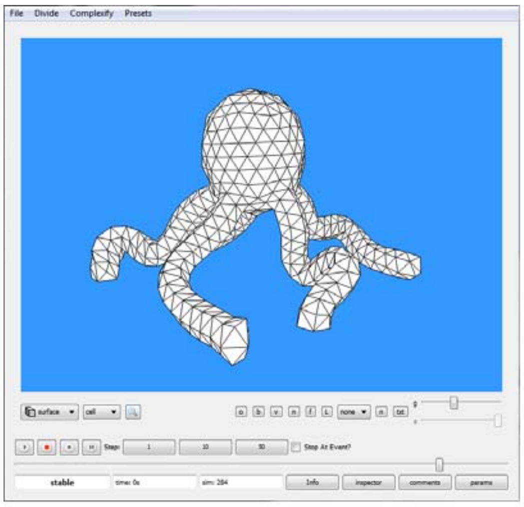
A non-interactive off-line simulator was built for larger simulations, and was complemented by a renderer which processed the simulation data and created vector images (.SVG format) for visualisation and analysis. The renderer was written as a custom Python script (using the Cairo vector graphics library[4]) and was used to generate many of the SDS2 images in this thesis. The animation *It Looks Like An Echinoderm* included in the DVD was also rendered using this system.

8.6.3 SDS3

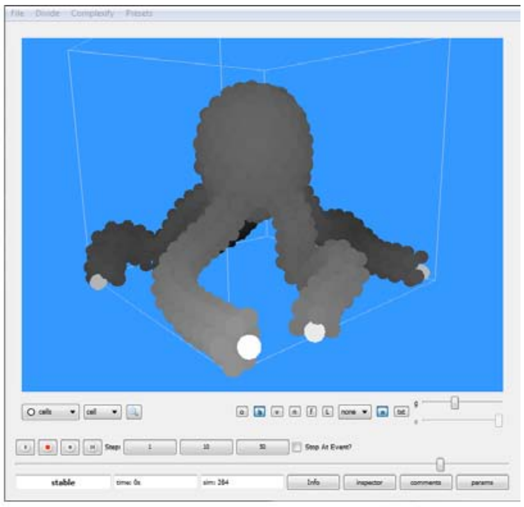
The SDS3 software is substantially more complex than SDS2 as it not only involved more complex data structures and algorithms, but, unlike the SDS2 simulator, it implemented the full SDS framework, including collision detection and simulation rewinding. The SDS3 simulator itself was built as a software module that is re-used in different tools.

A real-time interactive simulator was built to assist in exploring the different algorithms and growth models (Figure 8.4). It allows a simulation to be stepped through one frame at a time and provides a real-time visualisation of an s-morph's cells and geometric elements. It also allows the inspection of cells and geometric elements, so a user can, for example, see how much morphogen is inside a cell, or see an edge's desired length versus its current length. These features were crucial in designing, debugging and fine-tuning the algorithms and concepts of SDS3.

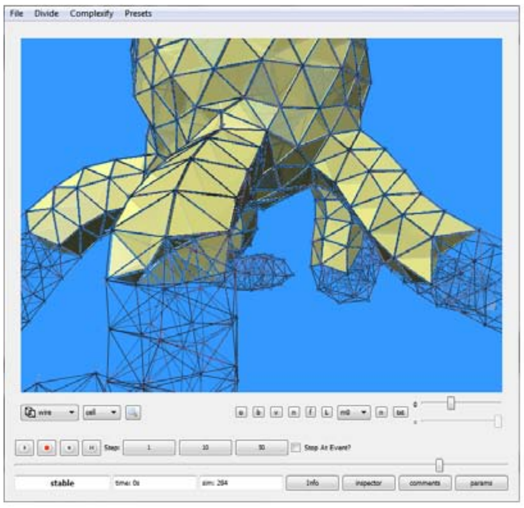
[4]`http://cairographics.org/`



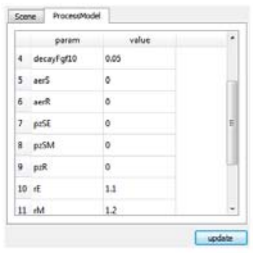
(a)



(b)



(c)



	param	value
4	decayFgf10	0.05
5	aerS	0
6	aerR	0
7	pzSE	0
8	pzSM	0
9	pzR	0
10	rE	1.1
11	rM	1.2

(d)

Figure 8.4: Screenshots of the interactive SDS3 simulator. (a) A simulation has been loaded. The lowest slider allows the individual frames to be cycled through. The other GUI elements allow various properties of the view to be changed. In this screenshot the unlit surface mesh of the s-morph is shown. (b) A cell view mode allows the individual cells to be visualised. The colour of the cells indicate their morphogen concentrations. In this example the growing tips are full of morphogen, and hence are coloured white. (c) Tetrahedra can be selected and shown or hidden. This is helpful when analysing sub-structures. (d) A secondary parameter window allows a user to adjust parameters of the simulation while it is running and instantly see the effects.

An off-line simulator was built to run all the lengthy simulations (typically overnight), and output the simulation data for later analysis. The input and output of this simulator occurs via a custom file format (§8.6.3.1). The output data can be imported either into a tool for interactively viewing the simulation (based on the real-time interactive simulator), or imported into Blender, a 3D modelling and animation package. The software interface between SDS and Blender is described later (§8.6.3.2).

At the time the system was built, there was a limited range of physics simulation software that supported tetrahedral meshes, including Idolib[5], VCG[6], and PhysX[7]. It wasn't clear if these systems could support tetrahedral meshes with a dynamic structure, and so the SDS3 physical simulator was built from scratch. In retrospect, the design of Idolib is nicely suited to a system like SDS, as it cleanly separates numerical integration technique from the mesh data structure, which allows different integration schemes to be swapped in and out. If the SDS3 system was to be used in a production environment, its simulator would need to be flexible and robust, and Idolib's design would allow for this. A large portion of this research was spent building and debugging the simulator and discovering how to integrate all the components. With this experience, a more stable and flexible version of SDS3 could be built in a fraction of the time using existing technologies.

8.6.3.1 Simulation File Format

During this research, the design of SDS3 was formative and changed on a regular basis. As features were being added or removed the simulators and tools were updated accordingly. In order for existing simulation data to be compatible with newer iterations of tools, a flexible file format was designed that was backwards and forwards compatible. Aspects of this format may be applicable to similar systems and so it is discussed here.

The SDS3 simulation file-format can be used in two different modes: to store the initial conditions for a simulation, or to store the results of a full simulation. When used as input to a simulator the file simply stores one frame containing the initial configuration of the s-morph. When used to store the output of a simulation, the file can store multiple frames of time-dependent data, for example, the changing structure of an s-morph. The simulation file is the primary channel of communication between simulators and tools. One software tool that was used early on in this research, for example, read in a full simulation file and, for each frame, extracted

[5]`http://idolib.sourceforge.net/`
[6]`http://vcg.sourceforge.net/`
[7]`http://www.nvidia.com/object/physx_new.html`

the surface mesh of the s-morph and converted it to the common `.ply` mesh format. These `.ply` files were then imported into Blender for rendering (before the Blender-SDS plug-in was developed (§8.6.3.2)).

A major design decision for this format was that it be both backwards and forwards compatible. Backwards compatibility allows older simulation files to be read by newer iterations of SDS. This allows new features to be added to the software without invalidating existing simulation data. The vertex freeze feature (§6.3.8), for example, was added quite late in the implementation, adding a new cell variable, c_{frozen}, to the simulators. If a tool reads an old file, that doesn't contain the vertex freeze information, then the variable is assigned a default value ($c_{frozen} =$ false). Forwards compatibility allows SDS tools to read simulation files generated by newer versions of the software. This is particularly useful as it means that new features (e.g., vertex freeze) can be added to the main SDS3 simulator and simulation file without breaking all the other tools for which the feature had *not* been implemented. The `.ply` surface mesh extractor tool, for example, doesn't care about whether cells are frozen or not, and so forwards compatibility ensures that it will only read the information from the simulation file it knows (or cares) about.

This compatibility of the format is possible through the use of simulation file *segments*. Segments are arbitrary chunks of frame-dependant data. There are segments for storing data such as the tetrahedral mesh, cell variables values, and morphogen values. New segments can be added easily via a simple API. If a simulation file contains a segment that a program doesn't understand then it can choose to simply ignore it. An additional advantage of this format is that tools can choose to selectively output features, useful for reducing the simulation file size. For very long animations, selective output is useful due to the large file sizes involved — it is much more efficient to output only the surface mesh of an s-morph and discard the other information.

When used to describe the set-up of a simulation, it was desired that the simulation parameters (gravity strength, time-step, stiffness coefficients, etc.) be editable by hand. Instead of designing a specific tool to do this, the simulation files are split into two parts: a human-readable component and a binary component. A simulation is described by two files, a plain-text `.cfg` file and an associated `.bin` binary file. The `.cfg` file specifies the simulation parameters and can be edited using a normal text editor. It also specifies the name, location and format of the binary data file associated with the simulation, which stores the data segments described in the paragraph above.

The file-format is described in full in Appendix C.

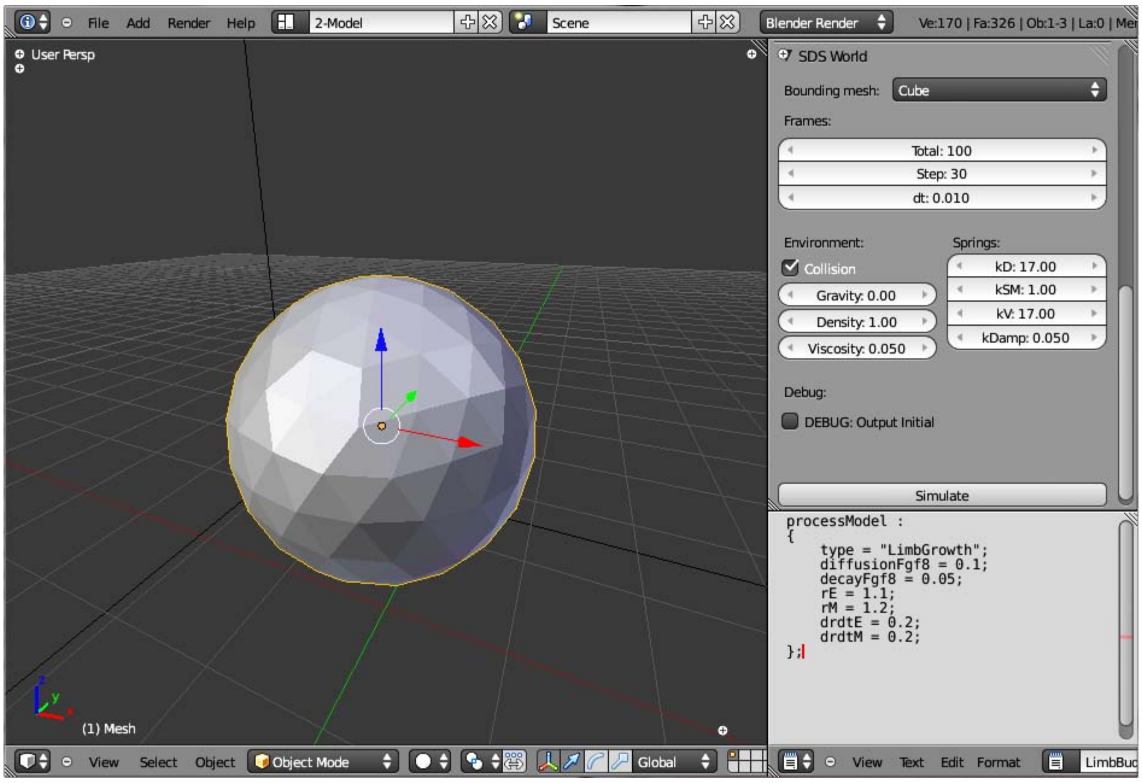
8.6.3.2 The SDS-Blender interface

To provide a more useful interface to SDS, software was written to allow SDS to interact with Blender, a popular open-source 3D modelling and animation package. This software was designed and implemented in collaboration with Bart Veldstra, as part of an undergraduate software engineering project. The SDS-Blender plug-in integrates SDS concepts directly into the primary Blender interface. A user can set-up an SDS simulation as a Blender scene, specify simulation parameters, specify parameters to a growth mode, and then press a *simulate* button (see Figure 8.5). The plug-in then exports the Blender scene and parameters to a format compatible with SDS simulation and runs the SDS simulation. The plug-in then imports the simulated scene as an animated Blender scene. Using this interface is far simpler than using the other tools. (Before this tool was developed, each separate step: simulation set-up, simulation, export to `.ply`, and import into Blender, was performed manually.)

Blender does not support tetrahedral meshes, and consequently s-morph information can only be specified at the surface level as a triangular surface mesh. To add an s-morph to the scene, a user builds a triangulated surface mesh (typically by adding a mesh primitive, such as an icosphere[8] or a cube), and then tags the mesh as an s-morph. When exported, the mesh is tetrahedralised (with a user-defined precision) and converted to an s-morph (as in §6.3.7). Morphogen concentrations of surface cells can be specified using Blender's vertex paint tool. Each colour channel of a vertex corresponds to a morphogen in SDS. For example, if a vertex is painted red (i.e., RGB $(1, 0, 0)$), the corresponding cell, c, will have $c_{\phi_1} = \mathrm{vol}(c)$, $c_{\phi_2} = 0$, and $c_{\phi_3} = 0$. Under this scheme, SDS simulations can only have three morphogens, but this could be extended to an unlimited amount using Blender's vertex colour layers, with which multiple layers of colours can be attributed to each vertex. Meshes can also be tagged as *static*. They will then be imported into SDS as static objects, with which an s-morph may collide.

The usability of SDS is dramatically increased when incorporated alongside other tools, such as Blender. When integrated within a more general 3D modelling package, SDS becomes a useful addition to the palette of 3D modelling and animation tools available. This observation holds for many of the generative systems available today. It is only through the successful integration of these systems within more general modelling tools, that the generative approach to modelling will enter the mainstream.

[8]An icosphere in Blender is a subdivided icosahedron.



Bounding mesh: Cube

Frames:

Total: 100
Step: 30
dt: 0.010

Environment:

✓ Collision
Gravity: 0.00
Density: 1.00
Viscosity: 0.050

Springs:

kD: 17.00
kSM: 1.00
kV: 17.00
kDamp: 0.050

Debug:

DEBUG: Output Initial

```
processModel :
{
    type = "LimbGrowth";
    diffusionFgf8 = 0.1;
    decayFgf8 = 0.05;
    rE = 1.1;
    rM = 1.2;
    drdtE = 0.2;
    drdtM = 0.2;
};
```

Figure 8.5: A screenshot of Blender showing some GUI elements for the SDS plug-in. The left panel shows a 3D view of a mesh, which will be converted to an s-morph. The top right panel allows the user to specify SDS simulation settings including the bounding mesh to use in the simulation, the number of frames and step size, whether to use collision detection, the gravity strength, and the spring stiffness coefficients. Pressing the "simulate" button then runs the SDS simulator and imports the animation back into Blender. The lower right panel is a text field that allows the user to specify the growth model to use, along with parameters. In this case, the user has chosen the "LimbGrowth" model (the basic limb model of Table 7.1), along with the morphogen diffusion and decay rates, division thresholds, and growth rates. Another panel (not shown) allows the user to select which meshes in the scene will be converted to s-morphs, and the precision of the tetrahedralisation process.

8.7 Summary

This chapter discussed various issues that arose during the development of SDS. A number of topics were covered, including: the emergence of structure, the trade-offs that exist between structural algorithms, the consequences of spatially embedding a form, how physical simulation plays a counterpoint to generative aspects of SDS, and why it is useful to consider biological models of growth. Aspects of the implementation of SDS in 2D and 3D were also discussed. Implementing and debugging SDS3 was, without doubt, the most time-consuming and difficult aspect of this project, and hopefully the discussion in this chapter will assist researchers who are building similar systems. The research presented here is just a first step towards full biological and physical simulation of organic form for computer graphics. The final chapter of this thesis proposes a number of issues that need to be addressed in order to achieve this goal.

Chapter 9

Conclusion and Future Work

This thesis presented a new system, the *Simplicial Developmental System*, capable of automatically generating a range of interesting organic forms, which are difficult, or impossible, to generate using existing methods. SDS presents a novel approach to form generation, based on the developmental and physical simulation of a volumetric structure embedded in an environment. It was demonstrated through a range of examples that SDS fulfils the objectives of this research outlined in Section 1.2.

This new technique of form generation was presented in a number of stages. The motivation and goals for this research were introduced in Chapter 1. A number of existing developmental systems were then reviewed in Chapter 2, highlighting features or deficiencies that inspired and drove the design of SDS. The general SDS framework was introduced in Chapter 3, describing each key component and its place within the framework: the Simplicial Morph, the growth model and cell program, the morphogen model, the structural transformations (cell division and cell movement), and the physical model. The details concerning the 2D and 3D implementations of each component were presented in Chapters 4 and 6. These chapters presented algorithms for modelling cell division and movement, soft-body physics, and the flow of morphogens on triangular and tetrahedral meshes. Experiments performed with the 2D model, exhibited in Chapter 5, demonstrated how SDS can be used to generate a limb structure and a morphogen stripe. These experiments were based on models of biological growth, and showed how local processes in SDS can be coordinated in order to form macro-scale structures. Lastly, Chapter 7 demonstrated the generation of limbed forms in 3D, using principles learned in the 2D experiments. These 3D experiments show the generation of continuous sequences of growing, complex, smooth, squishy, organic forms, which are very difficult to model using existing techniques. The combination of spatial interaction with a physical model supports a range of phenomena, including: prevention of geometric self-intersection

(Figure 7.16), shaping forms with external structures (Figure 7.6), and the inclusion of other environmental effects, such as attraction forces (Figure 7.14). The use of a volumetric representation (the tetrahedral mesh) in SDS allows internal forces to be modelled at a higher fidelity than existing systems, which typically only use a surface representation. In addition, the volumetric representation supports internal development events, which is also not possible in surface-based systems; for example, in the limb bud model presented (§5.1) it is the proliferation of *internal cells* that causes outward growth of limb-like forms.

The original, grand aim of the research presented in this thesis was to generate, through biological and physical simulation, the organic forms of Figure 1.1. The experimental results shown in Chapter 7 are simpler than these forms; but nonetheless demonstrate that the approach is worth pursuing further. SDS is still in its embryonic stage, requiring more development before maturing into a system capable of generating interesting complex structures such as those in Figure 1.1. This chapter concludes this thesis by offering suggestions for improvement to the robustness, expressiveness, and usability of SDS.

9.1 Growing Other Structures

A wide range of structures appear in organic forms, considering just the forms shown in Figure 1.1, for example, there are bulbs, tubes, tentacles, suckers, discs, tendrils, ridges, folds, clumps, hexagonal packings, branches, and stars. The Simplicial Developmental System was designed with these characteristics in mind; however, numerous technical and implementation problems meant that only a limited type of structure — the limb bud model — could be examined (in the 3D system) in the time available. Further research into the processes behind these other sub-forms is necessary in order to increase the expressiveness of SDS and improve its applicability as a general organic form modelling system.

In order to be widely adopted into mainstream 3D modelling, these processes would need to be encapsulated as, for example, parameterised modules, as was done with the limb bud module (Table 5.1). Such a high-level interface to SDS would build upon the wide range of different processes within SDS and offer a user a palette of different sub-forms and patterns to use within a design. The palette could consist of such sub-structures as tubular growths, rings, wrinkles, folds, and ridges; along with methods for coordinating the growth of these structures. For example, a *composition* method would allow a user to grow a ridge along a surface, and then cause the ridge to wrinkle. These methods operate at a low level by designing cell programs and morphogen patterns. Toward this goal, it would be highly beneficial to find a set of fundamental form and pattern generating processes, or *axioms*, upon which growth models can be built. One of these may be *directed proliferation*, in which a cluster of cells repeatedly divides and grows towards a target, causing the structure to stretch along an axis. This is vital for the growth of the limb forms shown in this thesis. Other axioms might include stripe formation, morphogen gradients, timer morphogens, and lateral inhibition.

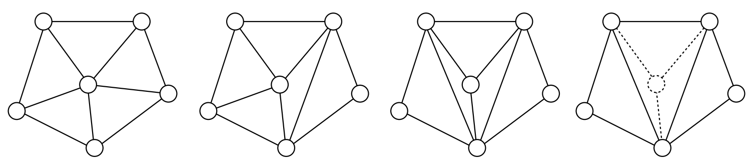
9.2 Transformations

The set of transformations presented in this thesis was necessary to achieve the results shown; however, it is just a small sample of the operations that can be performed on an s-morph. Future research could implement other operations and consider the implications for creativity and generative design. This section discusses a few of these operations, including cell death, surface fracturing, and adaptive meshing.

9.2.1 Cell Death

Cell death is an obvious exclusion from the transformations discussed in this research, particularly as it is vitally important in biological morphogenesis. The focus of this research was on the integration of structural transformations, physical simulation, morphogen simulation, and cell behaviour models, acting on an embedded triangular and tetrahedral structure. The biological limb bud model (§5.1.1) drove the development of the features of SDS. It required proliferating cells, and so cell division and cell growth were implemented. It required morphogen gradients, and so the morphogen model was developed. The limb bud model, in its primitive form, did not require cell death, and so this transformation was not considered. If more complex growth models are to be implemented, cell death will have to be incorporated into SDS.

Cell death can be implemented in SDS2 by following the approach of Duvdevani-Bar and Segel (1988). Their research extends Matela and Fletterick's triangulated graph model (which SDS2 is closely related to) with extra features, such as cell death. In their paper, cell death is described as the slowing down of activity and successive removal of connections to neighbouring cells. This is modelled on the triangulated graph as a sequence of edge-flips followed by deletion of a few elements (Figure 9.1a). Removal of boundary cells can be performed in a similar way.


(a) (left to right) Cell death in Duvdevani-Bar and Segel's model (Duvdevani-Bar and Segel, 1988). The inner cell elects to die and sequentially severs its adhesion to neighbouring cells. Once it has only three neighbours it is removed (the dotted lines) and the triangulation property is preserved.

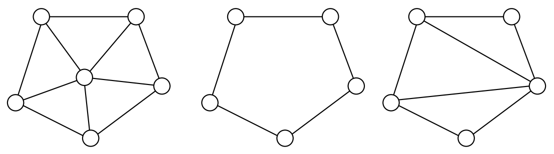

(b) Cell death with hull triangulation. (left) An internal cell chooses to divide. (middle) Remove the cell and all adjacent triangles. (right) Triangulate the hull. The triangulation can be governed by a metric, for example to preserve physical stability.

Figure 9.1: Two approaches for modelling cell death in a triangulated graph.

This method does not generalise to 3D as it requires edge-flips (see §6.5). An alternative method that does generalise, is to completely remove the structure surrounding a dying cell and then re-triangulate (or re-tetrahedralise) the empty hull (Figure 9.1b). This is a brute-force approach to cell death, similar in spirit to the balanced cell division algorithm (Algorithm 10).

One interpretation of cell death is as the *inverse* of cell division, where a dying cell is merged with an adjacent cell to create a *mother* cell. Using this interpretation, some cell division algorithms can be inverted to make cell death algorithms. A special case of the method shown in Figure 9.1b is illustrated in Figure 9.2, which is essentially the inverse of cell division via tetrahedral subdivision (Algorithm 9). Whether there exists a cell death operation that is efficient and results in nice structures is yet to be determined, and like cell division there are likely to be advantages and disadvantages of different methods.

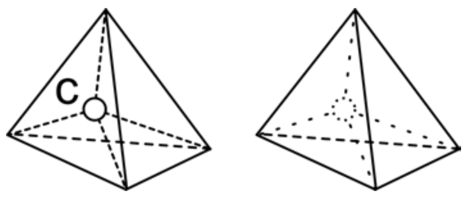


Figure 9.2: A simple example of cell death in SDS3. (left) A cell, c, with four neighbours elects to die. (right) The cell is removed and its four neighbouring tetrahedra are unified together to form the larger bounding tetrahedron.

Although technically challenging, a more critical consideration is the creative consequence of cell death. Is it necessary in order to achieve some types of structures? In some cases, like the formation of fingers (see §9.2.2), a cell death operation is useful, but not necessary. Finger-like structures can be grown either by dissolving the space between the fingers, or by growing the fingers individually like the growth of limb buds. So in this case, cell death seems unnecessary. One structural feature that seems to require cell death, or the removal of material, are small indents or holes in a structure.

Aside from generating interesting structures, cell death could support the simplification of structures. Cell division adds elements to an s-morph, which support the generation of complex structures. Likewise, cell death removes elements from an s-morph, and thus could be used to simplify a structure.

Cell death could be used to smooth bumpy surfaces that have too many cells, or to reduce geometric complexity while retaining visual features. Consider a large region of an s-morph, for example, that has many thousands of cells, all similar in behaviour, and clumped together. It may be possible that the behaviour and visual appearance of this cluster could be adequately modelled with much fewer cells. Cell death in this situation could allow many of the cells to die, while other cells grew to fill the empty space. This could result in a dramatic increase in the efficiency of a simulation, and is an example of the more general adaptive meshing transformations discussed below (§9.2.3).

9.2.2 Surface Fuse and Fracture

Cell death in the development of autopods (e.g., hands) allows the disintegration of webbing and separation of digits. The cell death model mentioned above fills in the space left by a dying cell. The other option is to allow empty space to remain, thus supporting material fracture. A surface fracture transformation could be designed to permit the deletion of cells, edges and simplexes, allowing the generation of complex surface features. One such transformation could detect when edges are under large strains and delete them if a threshold is reached (Figure 9.3). A similar operation could allow surfaces to fuse together (essentially the inverse of Figure 9.3). This would allow a growth model to change the surface topology of an s-morph. For example, an s-morph could start as a sphere and turn into a donut as shown in Figure 9.4. This feature could also support the modelling of biological phenomena like gastrulation and neurulation. The physics of surface fracture has been addressed in computer graphics models which may provide insight into a model suitable for SDS (Federl and Prusinkiewicz, 2004; Wicke et al., 2010).

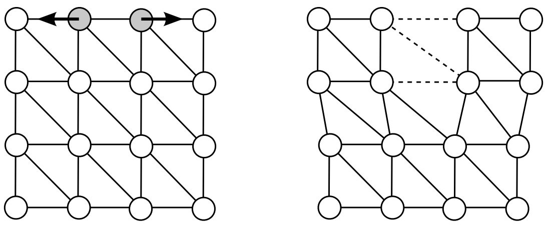


Figure 9.3: (left) A fracture operator could detect when large forces are pulling two cells apart, and (right) delete the edges that are strained beyond some specified threshold.

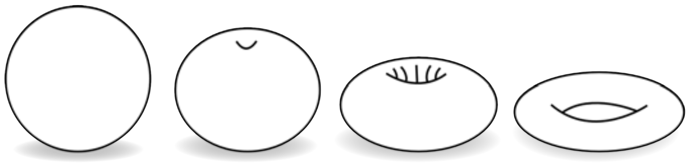


Figure 9.4: (left to right) Supporting changes in surface topology would allow, amongst other things, spheres to change into torii. (A topological change similar to the one shown here occurs during the process of gastrulation in early biological development.)

9.2.3 Adaptive Meshing

The mesh of an s-morph is adaptive and restructures itself to accommodate moving cells, helping to distribute stresses within the form, resulting in a more stable system and natural arrangements of cells and geometry. This adaptive nature of SDS can be extended in a number ways.

Within the current system, cell movement transformations only occur when cells cross over simplex boundaries. In some situations, it may be possible that great stresses build up in an s-morph without this geometric event occurring. An alternative approach could be to analyse the stresses occurring in an s-morph, and directly manipulate the structure of an s-morph to reduce the stresses. This could have the same effect as cell movements, but would be a more robust approach.

Adaptivity can also be used to simplify an s-morph, as proposed in the cell death section. If there are a number of similar cells or geometric elements that are grouped in a cluster, then efficiency gains could be made by replacing many elements with a few. An extreme example of this would be to replace a spherical cluster of cells with a single cell the size of the cluster. Whether the efficiency gains this transformation provides outweigh the computational cost of locating homogenous regions of the right shape is unknown. The reverse of this transformation could also be beneficial. Assume that an s-morph contains a region modelled with very few tetrahedra. If a complex morphogen pattern is required in that region, then the mesh resolution will need to be increased. A system could monitor whether morphogen patterns fulfil some user-specified precision, and if not, repeatedly subdivide tetrahedra in the vicinity of the low pattern resolution until the required resolution is achieved.

Adaptive meshing is a broad topic, and numerous mesh modifications are potentially suited to this application. Operations may act on a local scale (e.g., with "2-3" and "3-2" tetrahedral mesh operations Joe (1995)) or on a global scale (e.g., by subdividing an entire s-morph). Incorporating adaptive meshing into SDS would allow a user to directly specify the level of detail they would like, supporting a work-flow that ranges between fast *draft* simulations with coarse meshes to accurate *production* simulations with high resolution meshes.

9.3 Heterogenous Material

The simulations presented in this thesis have all been performed using a homogeneous material (all the spring stiffness coefficients are the same). As a result the forms are homogenously smooth and behave somewhat like marshmallows. One exciting avenue of research could be to explore the creative possibilities of allowing different material properties within the same s-morph. This would allow, for example, rigid bones, softer muscle mass, and a jelly-like material to be part of the same structure. A major challenge in modelling heterogeneous material is the problem of physical stability. Rigid material can be modelled in SDS using highly stiff springs, but this requires a large amount of computation time due to increased instability, a problem commonly observed in mass-spring simulation (Baraff and Witkin, 1992). Therefore alternative approaches must be sought. Research into real-time simulation of deformable bodies for surgical simulation may provide solutions (e.g., Lin et al., 2010).

In addition to simulation stability, another issue arises when considering the refinement of a heterogeneous material. For example, when a cell divides in a heterogeneous region, what would be an appropriate procedure for deciding the physical properties of the new edges and tetrahedra? Recent work into numerical coarsening may offer insight into a solution (e.g., Kharevych et al., 2009). In resolving these challenges, heterogeneous material would provide an extraordinary facet of physical realism to SDS.

9.4 Improving the Simulation Method

The choice of integration schemes for the physical and morphogen simulations was adequate to achieve the results presented in this thesis. However, these efficient methods fail to correctly conserve physical properties, such as the energy within an s-morph or the concentration of morphogens in cells. Moreover, they are typically unstable and without the appropriate choice of time step the physical simulation is prone to "blowing up". The use of a better integration technique, specifically for the soft body simulation, would improve the reliability of SDS and would allow the modelling of finer grained details.

The results of this thesis show forms with detail of roughly the same scale, the cells are all roughly the same size and mass. Figure 9.5 shows a typical example, in which there exists two cells in an s-morph that differ in mass by approximately five times, not a considerable difference in scale at all. In order to model structures with multi-scale detail, further development of SDS is necessary. Mass-spring systems do not perform well on structures that have widely different masses, as the step-size typically has to be decreased to remain stable on the smaller masses. This slows the simulation down considerably. One solution to this problem could be to design a method for updating the positions of the masses with a priority inversely proportional to their mass (for example, update smaller masses every time-step, and larger masses every N time steps). This would be simple to implement and would increase simulation efficiency; however, the accuracy and stability of such a method would likely be extremely difficult to assess and prove. Another solution might involve the modelling clusters of adjacent small masses with a single larger mass. The resolution of these issues will need to be addressed in future research in order to efficiently and robustly model structures that have interesting detail that varies over scale.

9.5 Morphogen Model

Morphogen creation, destruction, diffusion and decay plays the role of pattern creation within SDS. Distributions of morphogens can direct specific parts of a mesh to grow, for example, in the proliferation region of the limb bud model (§5.1), or in the stripe region of the Drosophila segmentation model (§5.2). Two classes of patterns were explored in this thesis (the gradient and the stripe) and there is great opportunity to investigate other pattern classes and their potential for form and texture generation.

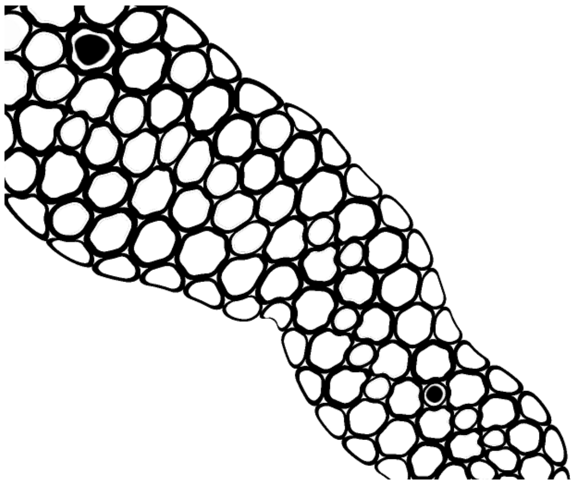


Figure 9.5: The representation in SDS can support systems of cells that drastically differ in size; however, the experiments shown in this thesis typically generate s-morphs which have the same size cells. This illustration shows one of the more extreme examples: a close-up of the form from Figure A.3 which shows two cells (marked), one of which is roughly five times the mass of the other.

Future research could examine the potential of a more sophisticated morphogen model based on reaction-diffusion equations (Turing, 1952). This would allow the generation of a wide range of self-organising patterns (such as those shown in Figure 2.11) which could be used to lay out geometric features or synthesize textures. This would be a challenging task as reaction-diffusion systems are notoriously difficult to control, and few studies have been made of the behaviour of these systems on dynamic surfaces (Leung and Berzins, 2003).

Another avenue of research is in the composition of morphogen patterns — taking two simple morphogen patterns and multiplying, adding or subtracting their cellular concentrations, resulting in a more complex pattern. Research could look into defining a set of axiomatic morphogen patterns (as in §9.1) and then designing a system for composing them. Composition could be done as described above, by directly multiplying morphogen values, or using more sophisticated means, such as compositional pattern-producing networks (Stanley, 2006).

9.6 User Friendliness

The design of an SDS form involves the specification of cell behaviours that coordinate dynamic morphogen patterns and geometric operations, i.e., the cell rule-sets. In addition to this, a number of parameters such as spring stiffness, viscosity, and morphogen decay rate need to be specified, many of which are abstract or irrelevant to the end-user. This interface to the user — with rule-sets and parameters — is not ideal, and becomes increasingly difficult to use as the design becomes more complex. If SDS and similar generative systems are to become accepted as design tools, the interface to the user should be as simple and intuitive as possible.

The parameter sets used to generate the results of this thesis where found manually, which is tedious and time-consuming. This is partially due to the indirectness of the parameters, but also due to the sheer number of parameters some of the growth models have. The first limb model (Table 5.1), for example, requires *nine* growth model parameters just to grow a simple lump, and this excludes the physical parameters! Therefore, an important contribution to improve the ease of use of SDS would be to reduce the number of parameters of the growth models.

Similarly, not only do the growth models need fewer parameters, they also need visually relevant and intuitive ones, i.e., parameters which directly control *visual aspects* of the generated form. Consider the simplicity (from the user's perspective) of a limb model that has parameters such as *limb size*, *growth speed*, and *limb length*. This would obviously be far simpler to use than the current model.

In SDS, not all sets of parameters (at least not in the limb bud model) result in successful growth (see Figure 5.9d for example). A mathematical analysis of the system could reveal the ranges of parameters that are successful; however, this analysis is likely to be extremely difficult due to the model's dependance on the temporal, structural and spatial properties of the s-morph. This difficulty is compounded because the effects of growth models are often dependent also on the *physical parameters* of the system.

If the parameter-space of a growth model cannot be reduced, then alternative methods for exploring the space may need to be examined. This could include the development of an interactive tool that allows users to drag parameter sliders and observe, in real-time, the effect they have upon a structure. This would require improvements to simulation efficiency (§9.4). Another approach is to use parameter-space exploration algorithms, such as evolutionary search techniques, which model parameter sets as genomes, and allow designs to be combined and evolved (Bentley and Corne, 2002; Sims, 1994). In order to build a simpler, "user proof" system, these issues would have to be considered in greater detail.

9.7 Summary

The Simplicial Developmental System is an entirely new approach for automated generation of organic volumetric forms, heretofore unexplored in a generative manner. The experiments presented in this thesis, drawn from real biological models of development, demonstrate that SDS is capable of modelling a variety of complex and smooth organic forms in two and three dimensions, but there is much more work yet to be done. This chapter examined many research paths for further development of SDS, and, in addressing these opportunities, progress will be made towards even greater biological and organic realism in 3D modelling for computer graphics.

Appendix A

Gallery of Additional Results

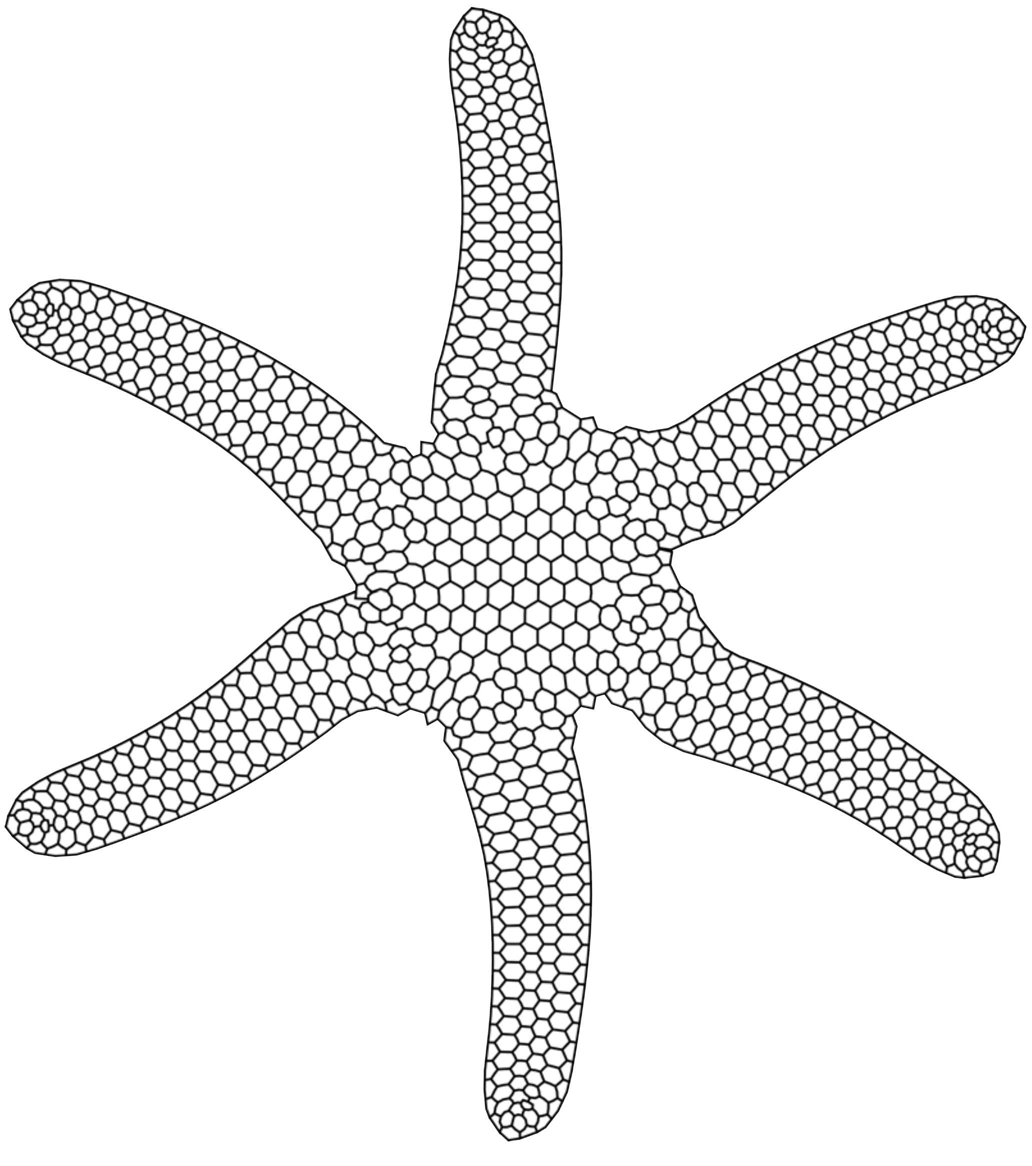


Figure A.1: A relative of the starfish shown in Figure 5.6.

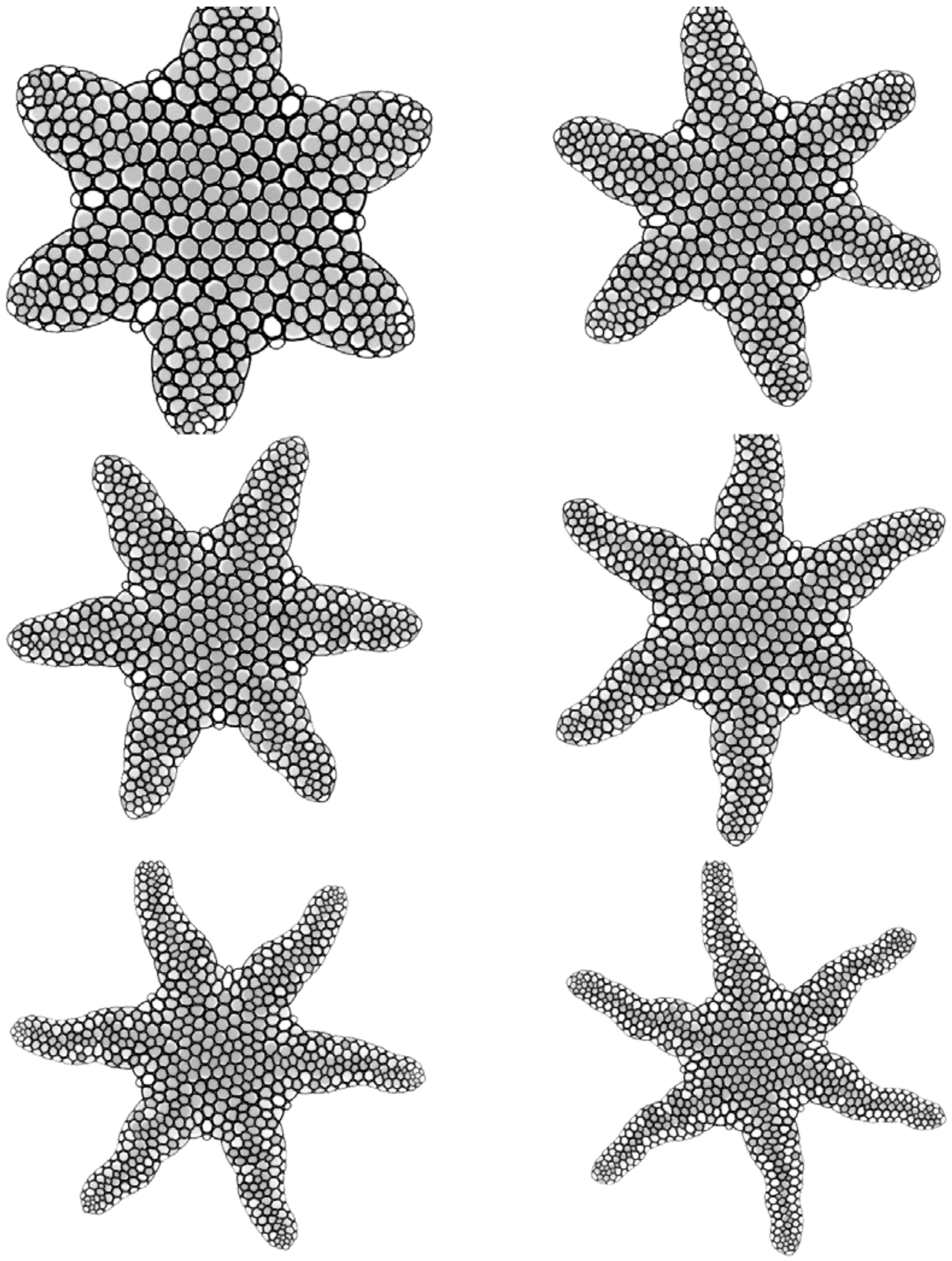


Figure A.2: (left to right, top to bottom) Frames from an animation created with the starfish developmental model. In this image the cells have been rendered as rounded polygons and shaded. Due to the increase in scale of the starfish over time, the image is zooming out to fit the entire form in frame.

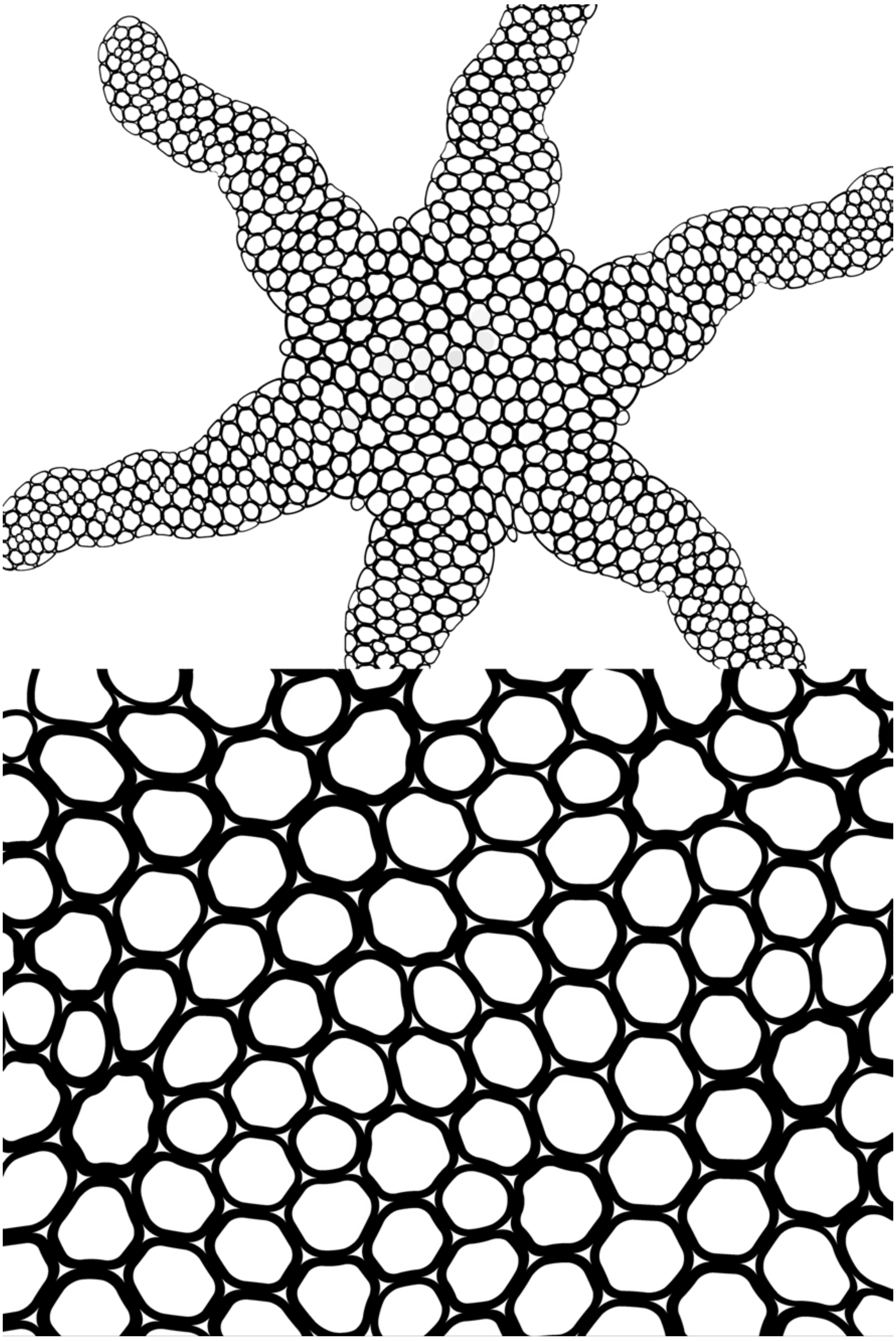


Figure A.3: Two close-ups of the form from Figure A.2. The lower image illustrates the organic arrangement of cells that emerges due to the minimisation of elastic energy in the s-morph.

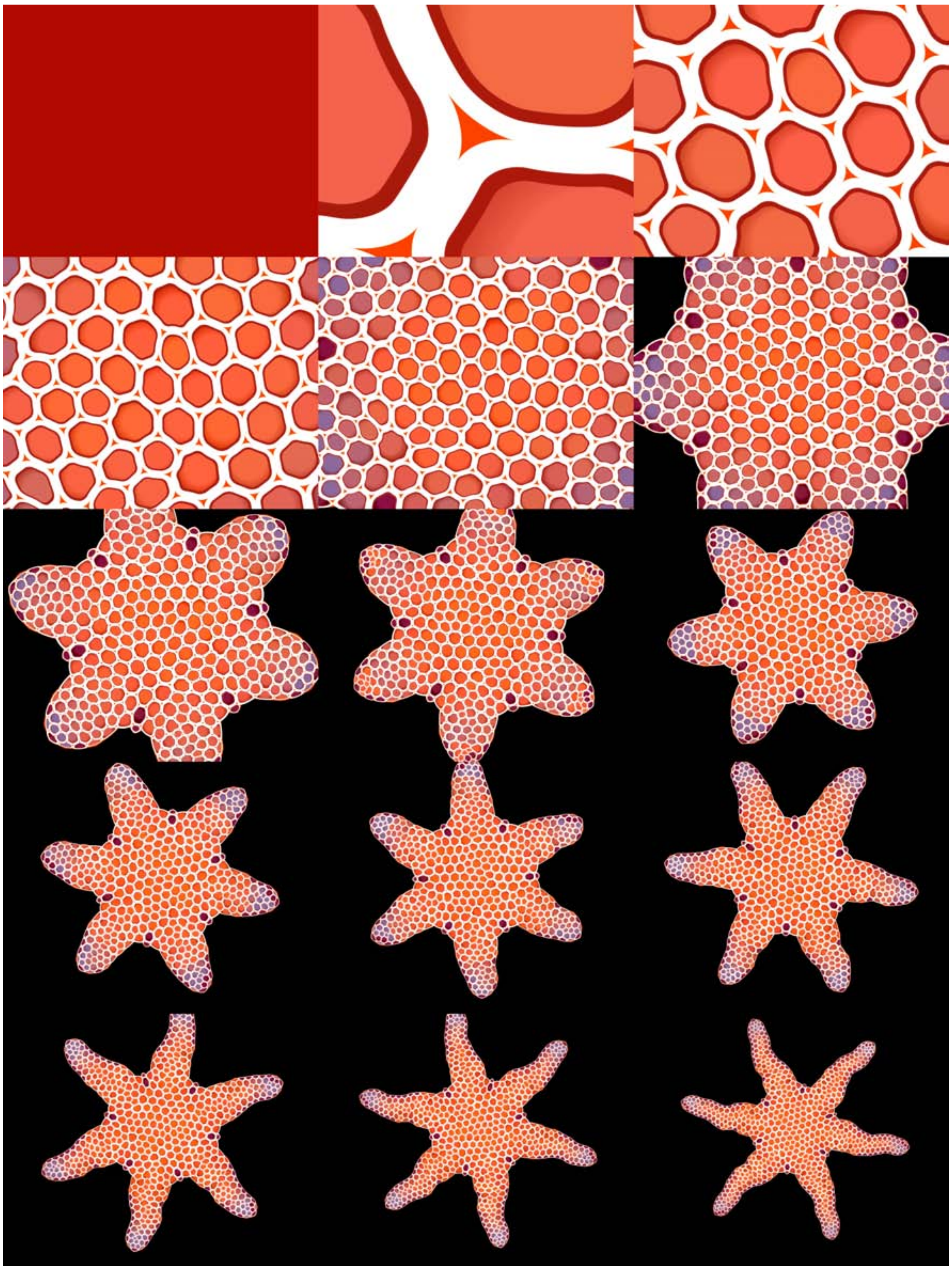


Figure A.4: (left to right, top to bottom) Frames from the animation *It Looks Like An Echinoderm*, created from a starfish growth simulation (see animation SDS2/6).

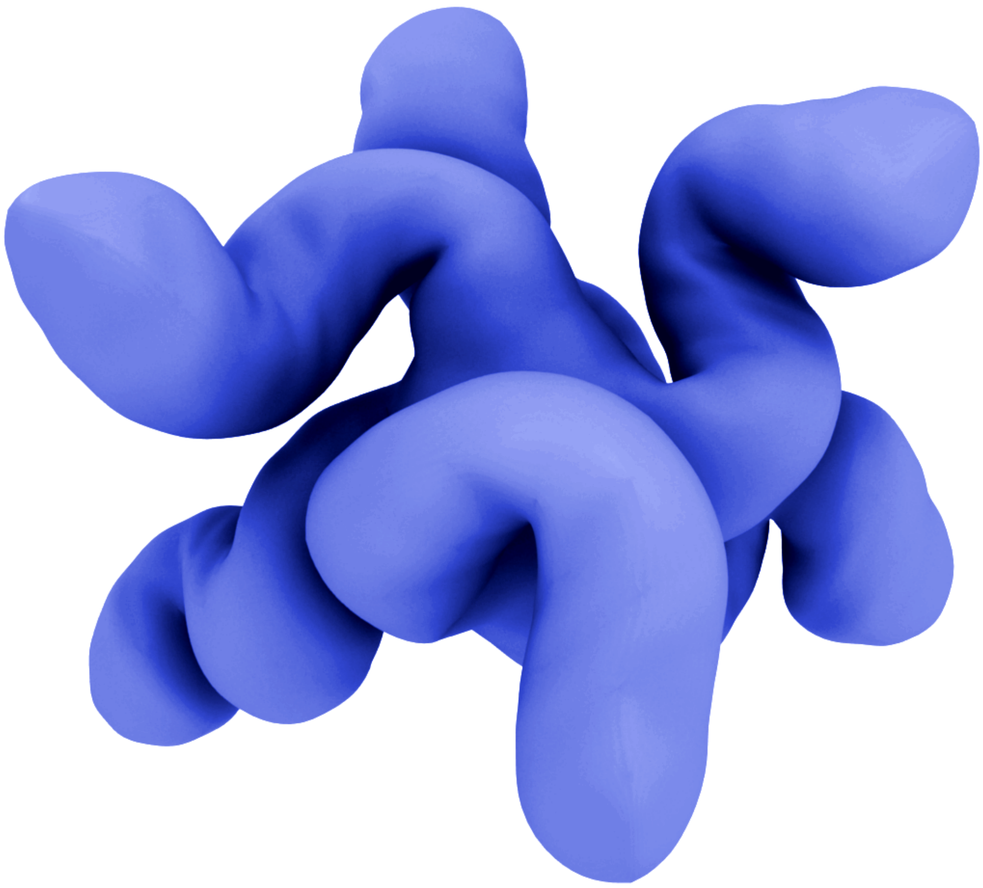


Figure A.5: A six-limbed form grown within a cube (cube not shown). This form was grown using parameters similar to the simulation shown in Figure 7.6.

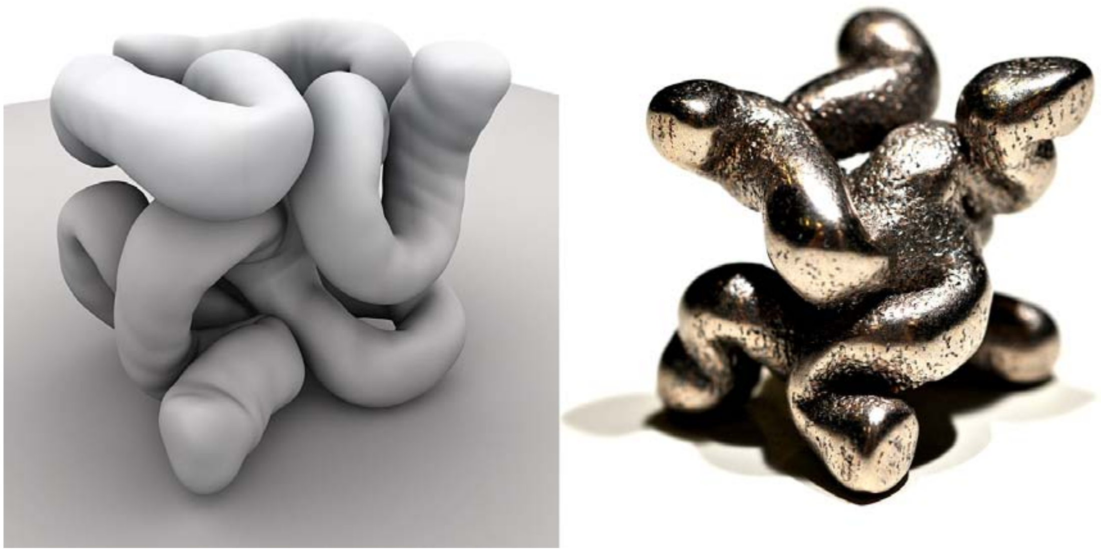


Figure A.6: (left) A form similar to the one above. (right) A photo of an SDS3 form manufactured in stainless steel.

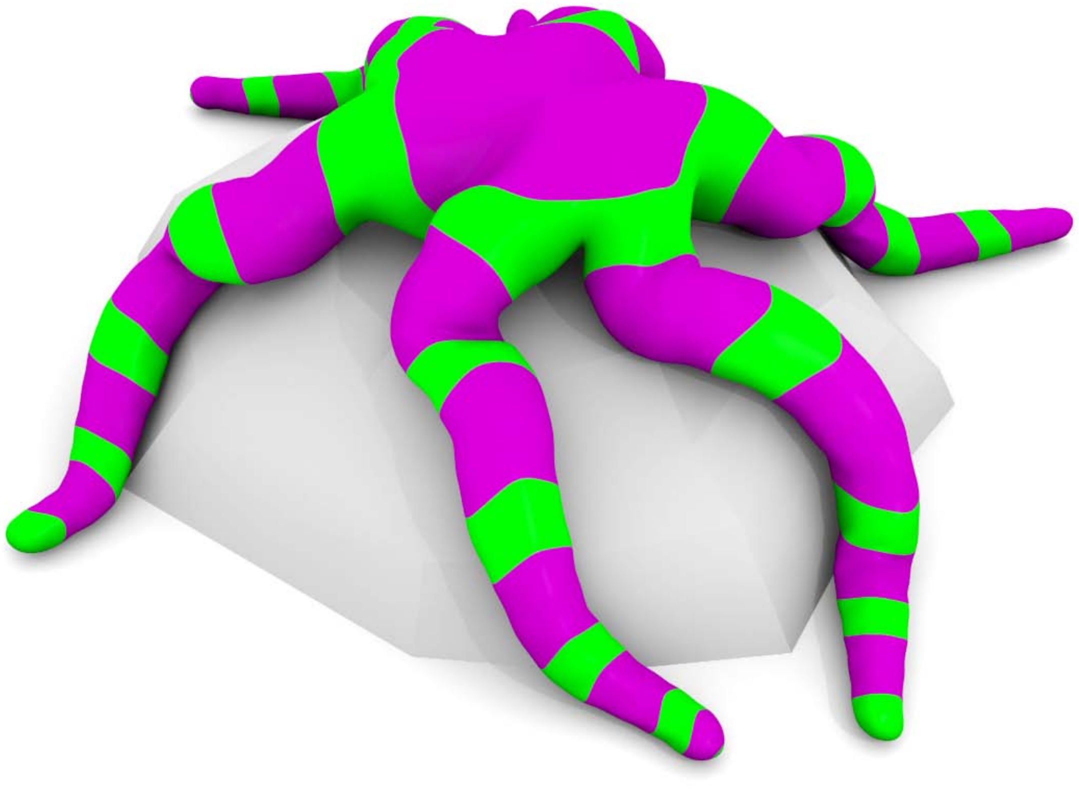


Figure A.7: A render of the last frame from the simulation shown in Figure 7.11, with the morphogens mapped to different colours.

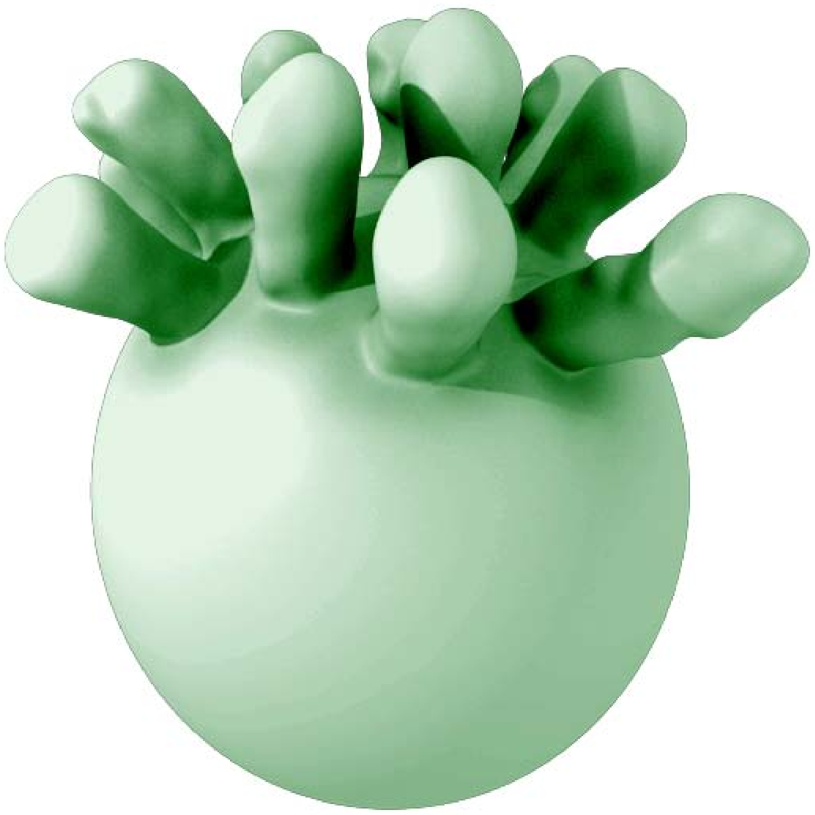


Figure A.8: A close-up of the budding simulation shown in Figure 7.12.

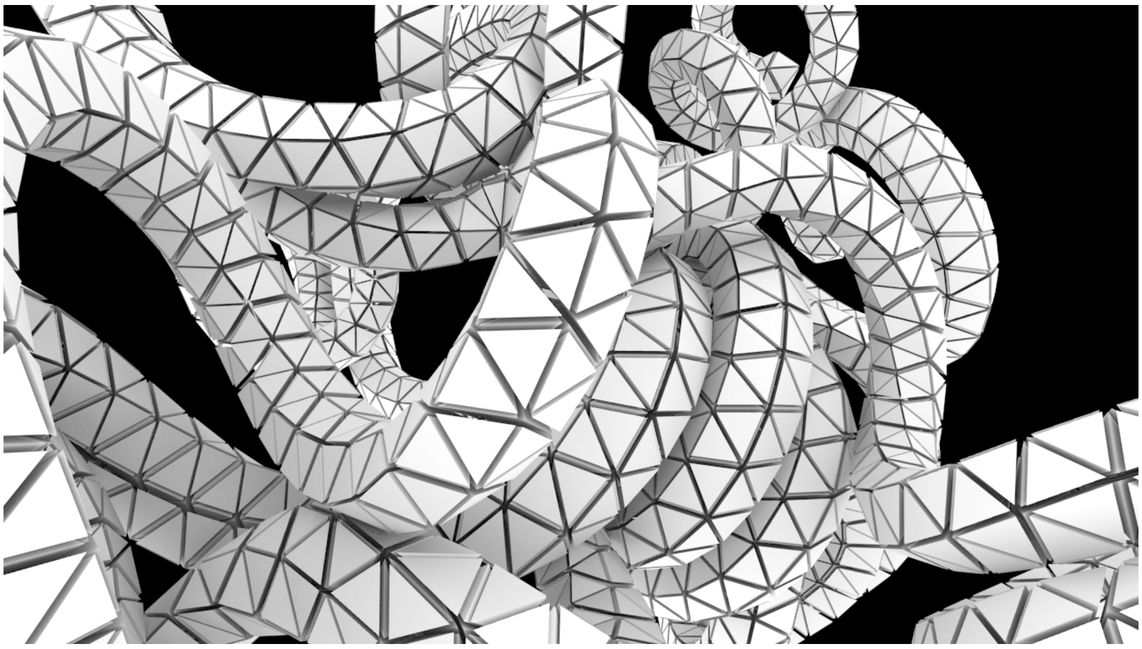


Figure A.9: A close-up of the simulation shown in Figure 7.16.

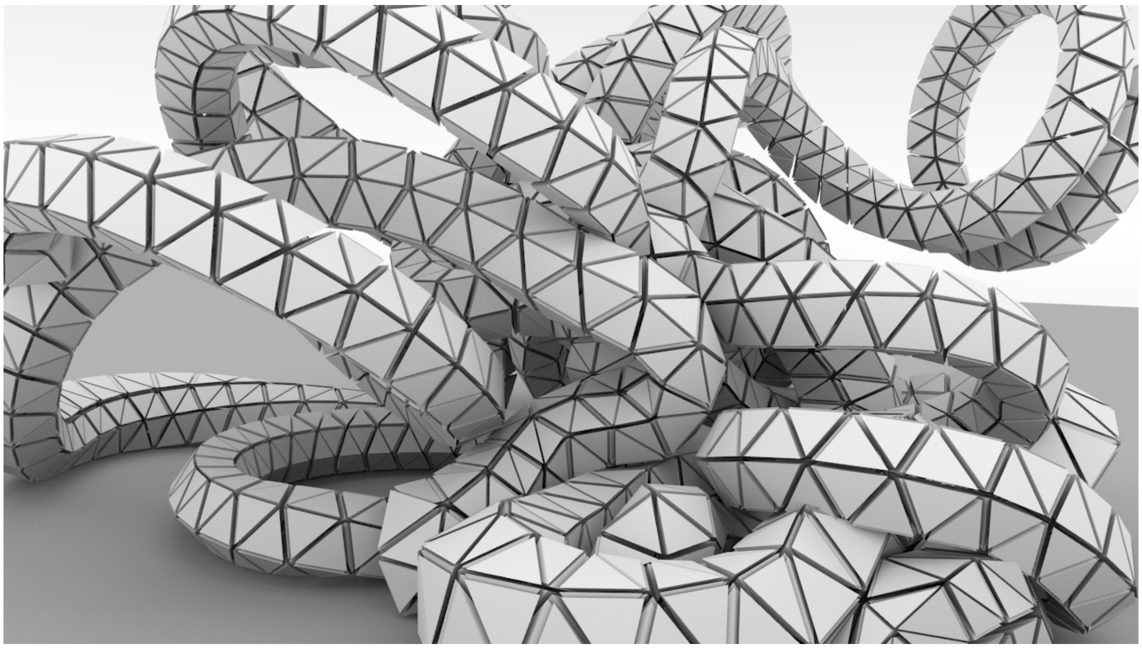


Figure A.10: A close-up of the simulation shown in Figure 7.16.

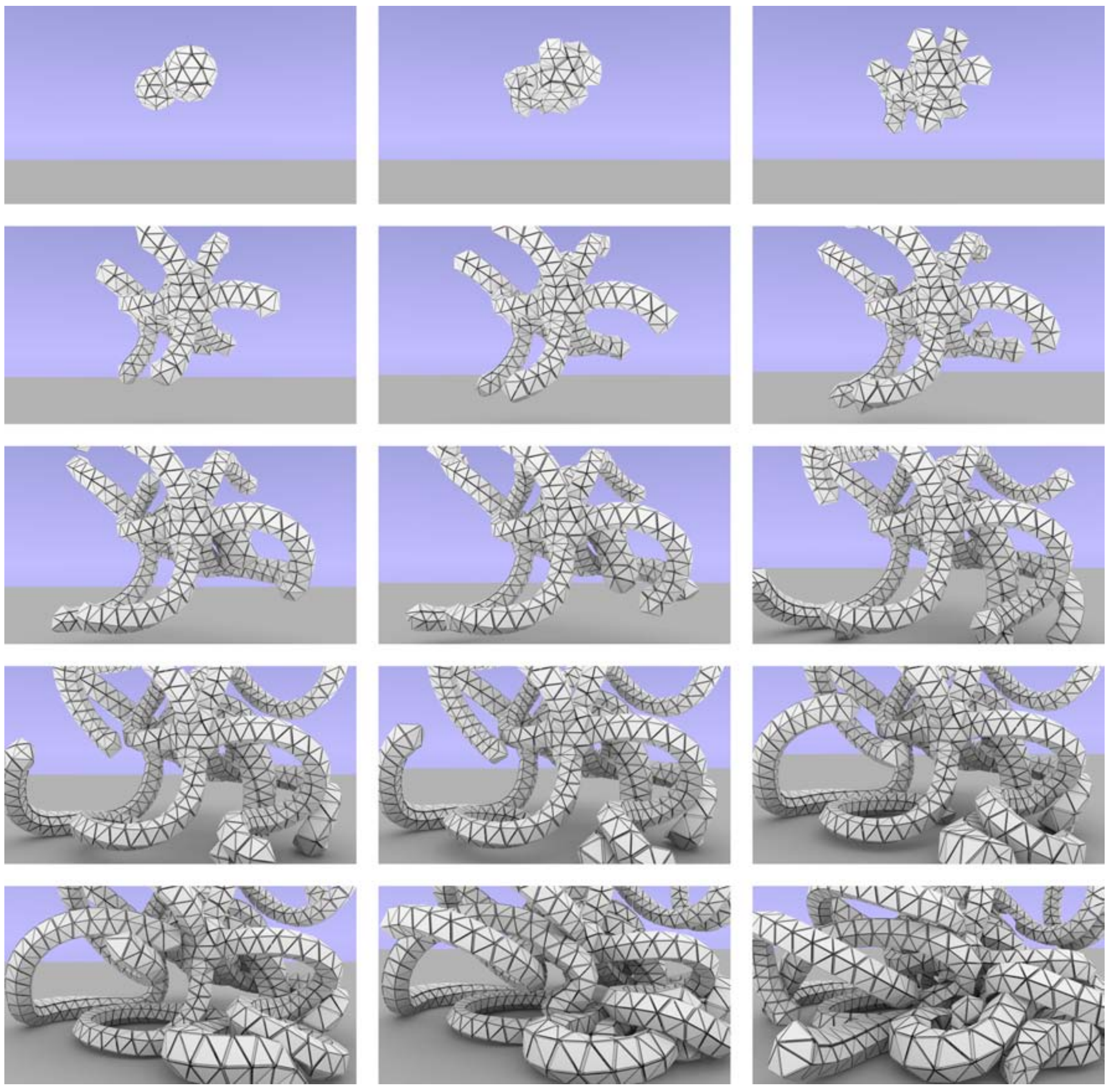


Figure A.11: A tetrahedral mesh view of the simulation shown in Figure 7.16.

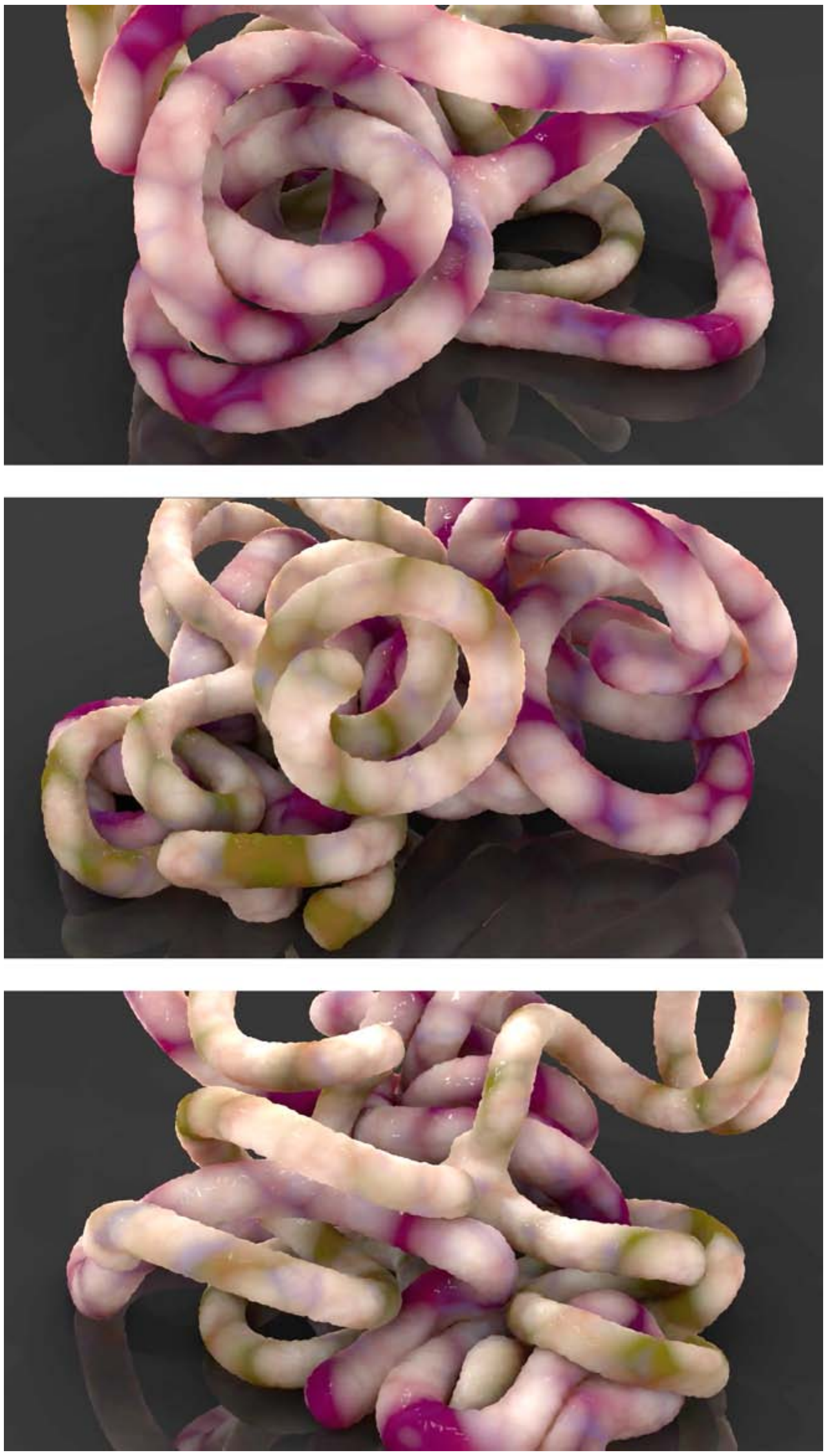


Figure A.12: Two intertwined tentacled s-morphs. This image shows three different view points. The form was generated from the simulation data shown in Figure 7.16. A frame was selected from the simulation and different textures were applied to the two smoothed meshes to distinguish them.

Appendix B

Modelling Cell Division with Separating Sets

The two SDS3 cell division algorithms introduced in this thesis lie at opposite ends of the spectrum — simplex subdivision (Algorithm 9 (p123)) is simple and efficient, but generates topologically asymmetrical configurations, whereas the balanced division method (Algorithm 10 (p125)) performs a brute-force re-tetrahedralisation, which is slower, but results in more physically stable configurations (as the resulting tetrahedra are more regular). In a typical simulation the cell division algorithm is likely to be executed many times, and so it is important that it is computationally efficient. The division algorithm used in the SDS3 software reduces the computational cost of the balanced division method by employing simplex subdivision under certain conditions (Algorithm 11 (p127)). This hybrid solution offers a balance between the two algorithms, however it is still inefficient because of the occasional brute-force re-tetrahedralisation it performs. It is desirable, then, to find a division algorithm that operates faster than brute-force re-tetrahedralisation but produces better quality results than simplex subdivision. This appendix offers a starting point for the design of such an algorithm, tentatively called *The Method of Separating Sets*. The algorithm is first discussed in the context of triangular meshes. These concepts are then generalised into 3D (tetrahedral meshes) and examples are given that perform division on surface and internal cells. Finally, a number of open problems concerning this approach are proposed for future investigation.

B.1 Separating Sets in 2D

Consider the SDS2 cell division algorithm illustrated in Figure B.1. The algorithm operates by first splitting the triangles surrounding a dividing cell (the *cluster*) into two halves. One half joins to one daughter cell, and the other half to the other daughter cell. The two halves can be found by choosing two edges (the *separating set*) which divides the cluster (shown in bold in Figure B.1b). The structure between the halves is then “filled in” by *stretching* the separating set along the edge joining the two daughter cells, in effect *transforming the edges into triangles.* This algorithm is simple, efficient, only generates a small number of new simplexes, and acts locally (i.e., structural modifications are confined to a local area).

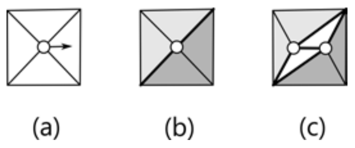


Figure B.1: A model of cell division in SDS2. (a) Given an internal cell and direction of division, (b) split the set of surrounding triangles into two halves (shaded). Between these halves lie two edges (bold). (c) Divide the cell in two along the direction of division, and join each daughter to each half. The final step is to add the triangles (white) between the two halves.

This approach can be formalised as the general cell division algorithm presented in Figure B.2. An important first step in this algorithm is to separate the triangles surrounding a cell into two groups and identify the boundary between the groups. This is done by splitting the face topology graph into two, in other words, finding a *minimal* set of edges whose removal would separate the fore and aft nodes. This set of edges (the separating set) is a key component of this algorithm. The goal, once a separating set has been found, is to add a vertex and change the mesh in such a way that, from the point of view of each half the structure has not changed at all. Figure B.2d shows the separating set, and Figure B.2e shows such a change. From the point of view of triangles a, b, c and d, the structure hasn’t changed, the two halves $\{a, b\}$ and $\{c, d\}$ still “think” they are joined to the original separating edges, this is why the transformation acts locally.

A crucial problem that arises in this algorithm is that not all separating sets can be stretched along a given direction (Figure B.3). For a given separating set the division direction has to lie within an area defined by the separating edges (e.g., Figure B.3e). Therefore, not only does a separating set have to be found, it has to be *valid* too. Finding this valid separating set it the most demanding part of this algorithm. This is fairly simple in 2D, as triangular clusters always form a “fan” around a vertex (corresponding to a *cycle* face topology graph), and therefore

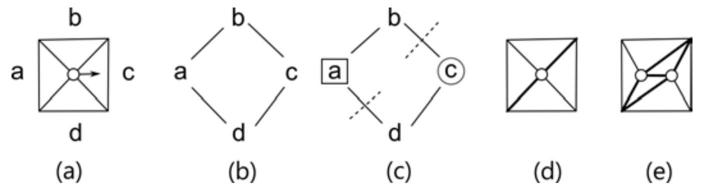


Figure B.2: Dividing a cell in SDS2 by stretching a separating set of edges. (a) A cell surrounded by four faces (triangles), a, b, c, and d elects to divide in the direction shown. (b) Construct a *face topology graph* (where faces are represented by nodes and edges connect nodes if two faces in the mesh share an edge). (c) Choose one face (in the direction of division) to be the *fore* face (circled) and one face (the greatest distance away from the fore node) to be the *aft* face (squared). Cut the topology graph into two pieces, one containing the *fore* face and the other containing the *aft* face. The dashed lines represent the cut. (d) The cut corresponds to edges in the s-morph which divide the face cluster into two parts. (e) Finally, split the cell into two, and *stretch* the two dividing edges along the edge connecting the daughter cells, thus forming two new faces.

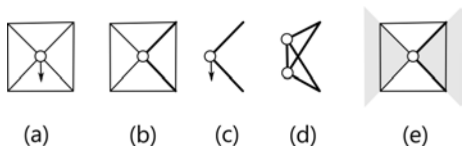


Figure B.3: An example of an invalid separating set. (a) A cell and its direction of division. (b) The bold edges have been selected as a separating set. (c) The separating set is then stretched in the direction shown, resulting in (d) two triangles which intersect, an invalid configuration in SDS. (e) This separating set can only be stretched in a direction that lies within the shaded region.

a separating set is always composed of just two edges. In 3D, however, the sheer number of different tetrahedral arrangements that can occur around a single vertex makes this task extremely difficult. This is discussed in the next section.

B.2 Separating Sets in 3D

This section considers how the method of separating sets can be performed on a tetrahedral mesh. It is simplest first to study the case of a surface cell. Figure B.4 presents a simple example of finding a separating set of faces for a surface cell. For any given configuration of tetrahedra there may be a number of separating sets. After finding a separating set it can be stretched (Figure B.5). It may be tempting to compare the configuration of surface triangles to the 2D case shown in Figure B.2, but this is misleading — the tetrahedral structure under the surface may be more complex than in Figure B.4. Figure B.6 gives an example of a cluster where the surface consists of the same four triangles as in Figure B.4, but the tetrahedra underneath are arranged differently, leading to a different set of separating sets.

Even when restricting our attention to surface cells, the range of different tetrahedral clusters is large. A first step for developing this research further would be to enumerate the different tetrahedra topology graphs that arise around a surface cell. Knowing all the possible graphs will be helpful when designing an algorithm for finding valid separating sets. Investigation with these structures leads me to believe that, with a little effort, the surface clusters should be easily enumerable. Enumerating the internal clusters, however, will pose a difficult challenge.

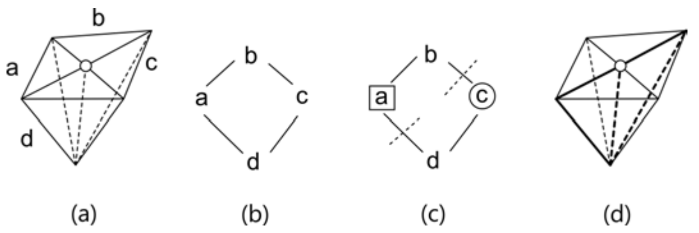


Figure B.4: Finding a separating set for a surface cell in SDS3. (a) A surface cell connected to four tetrahedra, *a*, *b*, *c*, and *d*. (b) The tetrahedra topology graph (in which tetrahedra are represented as nodes, and faces that lie between tetrahedra are represented as edges). Note that the graph is identical to Figure B.2b. (c) The fore and aft nodes have been selected, and a separating set of faces found. (d) The separating set (bold) splits the tetrahedral cluster into two halves.

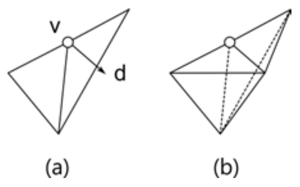


Figure B.5: Stretching a separating set of faces for a surface cell. (a) A cell, *v*, connected to the separating set of faces chosen in Figure B.4d, divides in direction *d*. (b) The faces are stretched and become tetrahedra.

Finding separating sets for an internal cluster is demonstrated in Figure B.7. Once found, an internal separating set can be stretched (Figure B.8). This example is one of the simplest internal clusters, and yet it has a fairly complex topology graph and a number of potential separating sets. As happens with triangular meshes, there is much freedom in selecting a separating set, but not all separating sets are valid — the direction of division, *d*, occasionally conflicts with the choice of separating set. If a separating set is discovered and found to be invalid, there are two options: find another separating set which is valid with respect to *d*, or change *d* so that the separating set becomes valid. Finding a separating set that fits a division direction may be computationally demanding, or even impossible. On the other hand, adjusting *d* to suit a particular separating set inhibits the directional freedom of a dividing cell. Which method is appropriate will depend on the quality required and the efficiency of the algorithm that finds separating sets. A brief rumination on what other tetrahedral clusters arise internally will reveal the difficulty involved in

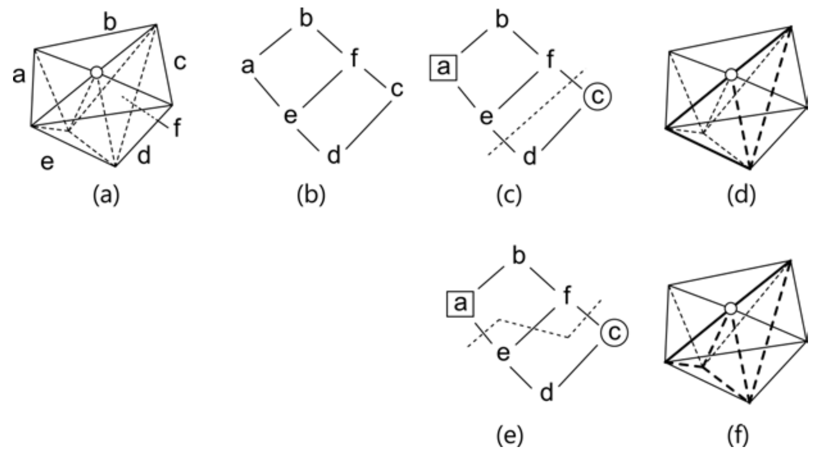


Figure B.6: A complex tetrahedral cluster of a surface cell. (a) A surface cell connected to six tetrahedra, $a \ldots f$. (b) The tetrahedra topology graph. (c,d) A separating set and its corresponding faces. (e,f) An alternative separating set and its corresponding faces.

enumerating them all. A complete enumeration may not be necessary in order to develop an efficient algorithm, but it is likely that it would greatly assist.

One final observation of the example given in Figure B.7, is that stretching each of the separating sets shown gives results identical to performing a simplex subdivision. For example, stretching the separating set in Figure B.7d corresponds to a tetrahedral subdivision, Figure B.7f to an edge subdivision, and Figure B.7h to a face subdivision. An obvious question then arises: *are all separating set divisions equivalent to simplex subdivisions?* If they are, then this method needn't be explored further; however, if this method turns out to be more expressive than simplex subdivision then it is worth investigating further.

B.3 Further Work

There is much work to be done in order to develop the method proposed here into a fully functional cell division algorithm for SDS3. There are a number of questions which need to be answered if this approach is to be fully explored:

- Given a tetrahedral cluster, how hard is it to find a valid separating set?
- Is it easy to find a minimal separating set?
- Given a separating set and a direction, d, is it always possible to modify d such that the set is valid?
- What constitutes a "good" separating set? (e.g., number of elements, regularity of triangles, planar alignment?)

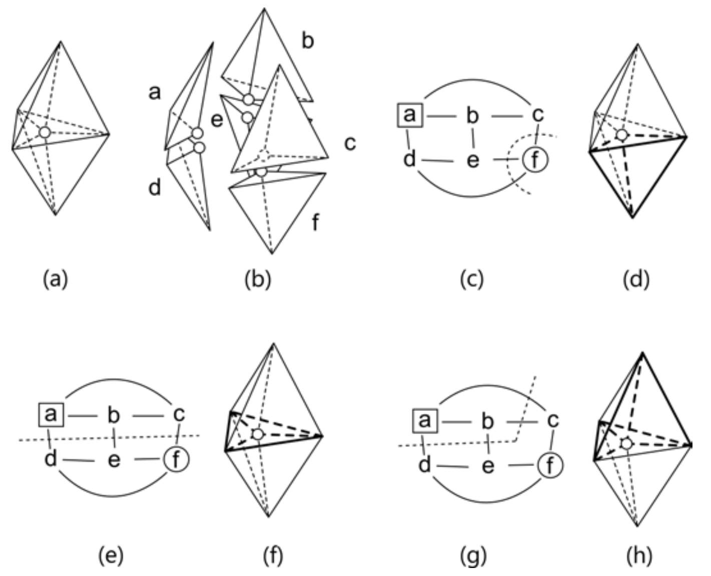


Figure B.7: Some separating sets of an internal cell. (a) A cell, surrounded by six tetrahedra, $a \dots f$, as labelled in (b) the exploded view. (c) The tetrahedra topology graph. Tetrahedron f has been selected as the *fore* tetrahedron and a as the *aft*. A separating set consisting of all edges connected to f has been chosen, corresponding to (d) the faces highlighted in this figure. (e,f) and (g,h) illustrate alternative separating sets.

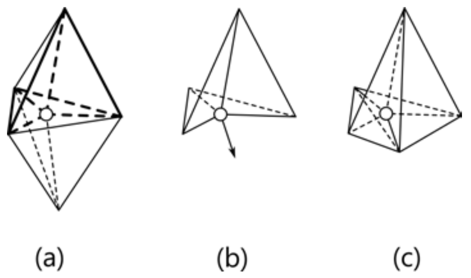


Figure B.8: Stretching an internal separating set. (a) The separating set from Figure B.7h. (b) The four separating faces along with the intended direction of division. (c) Stretching the faces along the direction of division converts them into four new tetrahedra.

- Can we discover the structure of all tetrahedra topology graphs?
- Can you efficiently find a separating set, for example, using dynamic programming?

Although not explored in full here, the method of separating sets may provide an entry-point into developing a better, more efficient algorithm for modelling cell division in tetrahedral meshes.

Appendix C

The SDS3 Simulation File-Format

A simulation in SDS3 is stored in two files, a human-readable `.cfg` file which contains information about the entire simulation (e.g., values of parameters), and a binary `.bin` file which contains time-dependent frame-specific data (e.g., the structure of an s-morph). This appendix presents the formats of these two files, in order to complement the discussion in §8.6.3.1 and provide insight into the technicalities of simulation serialisation in SDS3 in its constantly evolving state.

Configuration file The configuration file for an SDS3 simulation is stored in a `.cfg` file, in a format compatible with *libconfig*[1], a library for processing structured configuration files. This format was chosen due to its simplicity over other human-readable formats, such as XML, and the ease-of-use of the library's API. An example simulation setup configuration file is shown below. The C-style comments (//) describe each parameter.

Note that the parameter `numFrames` stores the number of frames in the simulation data. If `numFrames` is equal to 1 then this indicates that the binary file only stores one frame, and is probably intended as initial conditions for a simulation. If `numFrames` equals -1, then the number of frames in the simulation file is unknown, and the simulator or tool has to count the number of frames in the binary file (see below).

```
simulation : {
  dt = 0.01; // Step-size for simulation
  numFrames = 1; // Number of frames in the simulation data (see above)
  collisionInterval = 1; // Check for collisions every N frames
  gravity = 0.0; // Strength of the gravitational force
  kD = 17.0; // Stiffness coefficient for edge springs
  kSM = 1.0; // Strength of surface springs (proportional to kD)
  kV = 17.0; // Stiffness coefficient for tetrahedron springs
```

[1]`http://www.hyperrealm.com/libconfig/`

```
  kDamp = 0.05; // Damping coefficient for springs
  viscosity = 0.05; // Viscosity of environment
  framedata = "out.bin"; // File name of binary data
  time = "2011-Feb-18_12:34:43"; // Time this file was generated
  worldInfo :  { // Contains extra information about the world
    bounds : { // Specify the center and size of the world bounds
      x = -64.7145462; y = -49.44396782; z = -64.12328339;
      dx = 129.42908859; dy = 122.48793602; dz = 128.24655914;
    };
  };
  processModel :  { // Specifies the process/growth model
    type = "CurlingB"; // Curling limb program
    diffusion = 0.1; // Diffusion rate of the primary morphogen
    decay = 0.05; // Decay rate of the primary morphogen
    rE = 1.1; // Division threshold of surface (epithelium) cells
    rM = 1.2; // Division threshold of internal (mesenchymal) cells
    drdtE = 0.2; // Growth rate of stimulated epithelium
    drdtM = 0.2; // Growth rate of stimulated mesenchyme
    curlingFactor = 3.0; // Curling amount of the limb
  };
  frameSpecification = ( // Describes format of a frame in .bin file
    { type = "mesh"; version = "19042010"; }, // The tetrahedral mesh
    { type = "organism"; }, // Organism information
    { type = "processinfo"; } // Process/growth model information
  );
};
```

Binary file The second part of a serialised simulation is the binary file. It was decided that, due to the large amounts of simulation data that can be generated, storing time-dependent data in binary format, instead of text format, would save a lot of storage space. The binary file may contain a number of frames of simulation data. The exact structure of a frame in a binary file is described in the associated config file. The listing above, for example, specifies that each frame contains three segments: the *mesh*, *organism*, and *processinfo* segments. A simulation binary data file (`.bin` file) is structured as follows:

```
uint MAGICNUMBER  // used to check that this is an SDS3 simulation file
uint VERSION  // major version of the file-format
[Frame1]
[Frame2]
[Frame3]
...
```

During operation, the simulator outputs each frame as it is generated, appending it to the simulation file. The simulator was in constant development and, as such, the code was volatile, often resulting in the simulator crashing after a period of simulation. The simple layout of the binary file allows the simulator to crash without corrupting existing frame data. This greatly assists in debugging crashed simulations; however, one side effect of this simple appending layout is that no frame count can be stored at the start of the file. If the simulator completes a simulation it will write the frame count into the config file (the `numFrames` variable), but if it crashes, then `numFrames` will equal −1 and the number of frames is unknown. If this occurs then any program loading the file has to step through each frame and count the total.

The structure of an individual frame is as follows:

```
uint sizeOfFrameInBytes  // total size (in bytes) of all segments
uint frameNumber  // number of this frame
double currentSimulationTime  // time of this frame
uint numberOfStepsTaken  // number of simulation steps taken so far
[Segment1]
[Segment2]
[Segment3]
...
```

The order and type of segments in a frame are specified in the configuration file. In general, a segment is simply a chunk of binary data, with an unsigned long at the start that specifies its size (so a program can quickly skip over it if needed).

```
ulong sizeInBytesOfThisSegment
[SegmentData]
```

The segments can describe any elements of an SDS3 simulation that change over time. In this research, three segments were primarily used, the *mesh* segment which describes the structure of the tetrahedral mesh, the *organism* segment which describes the properties of the cells, and the *processinfo* segment, which stores process model dependent information, like custom cell variables and morphogen values. These segments are structured as follows:

Mesh Segment

```
uint numverts // number of vertices
uint numedges // number of edges
uint numouterfaces // number of outer faces
uint numtetras // number of tetrahedra
AABB bounds // smallest bounding box which fits this mesh
forall vertices v:
    vector3 v.x // current position
    vector3 v.ox // last position
    vector3 v.f // force acting on this vertex
```

```
        double mMass // mass
forall edges e:
        uint index(e.v(0)), index(e.v(1)) // index of two endpoints
        double e.rest // rest value
        double e.kD // stiffness coefficient
forall faces f:
        uint index(f.v(0..2)) // index of three vertices
        double f.rest // rest value (unused)
forall tetras t:
        uint index(t.v(0..3)) // index of four vertices
        int index(t.n(0..3)) // index of neighbouring tetrahedra
        double t.kD // stiffness coefficient
        double t.rest // rest vale
```

Organism Segment

```
uint numberOfCells
forall cells c:
        uint index(c.v) // index of associated vertex in mesh segment
        double c.r // radius
        double c.drdt   // rate of growth
```

Example ProcessInfo Segment

```
uint numberOfCells // number of cells (used for sanity check)
forall cells c:
        double c.morphogens[0]
        double c.morphogens[1]
        double c.customCellVariable1
        int c.customCellVariable2
        ...
```

As new features are added to the system, new segments can be added to the file-format to store associated time-dependent data. In conclusion, this modular file-format supported the two year development of a system which was constantly being changed, whilst maintaining compatibility with old simulation data-sets and across different implementations of tools.

Appendix D

Expanding the Tetrahedron Force Equation

Implementing the physical simulator in code (procedurally) requires the expanded form of Equation 6.16. The expansions for the edge forces can be found by hand, but the expansion for the tetrahedron forces is significantly more involved. It is easiest to expand computationally using a symbolic manipulation tool such as Mathematica[1]. For the sake of completeness, the Mathematica code for generating the tetrahedron force equations is presented in this appendix.

A necessary preliminary is to first define the gradient operator, which is used to compute the $\frac{\partial C}{\partial c_x}$ terms:

```
Clear[Grad];
Grad[f_, vars_List] := First@Outer[D, {f}, vars];
```

The volume, V, and constraint value, C, of a tetrahedron with vertices at positions a,b,c, and d, with a given rest value is:

```
VolTet[a_,b_,c_,d_]:=(1/6) * ((b-a).Cross[c-a, d-a]);
CTet[a_,b_,c_,d_,rest_]:=(VolTet[a,b,c,d]-rest)/rest;
```

The damping term, $s_{k_{damp}} \sum_{\tilde{c} \in s} \frac{\partial C(s)}{\partial \tilde{c_x}} \tilde{c_v}$, can be computed as:

```
TetDamping[ax_,bx_,cx_,dx_,av_,bv_,cv_,dv_,rest_,skdamp_] :=
  Table[skdamp*Grad[
          CTet[ax, bx, cx, dx, rest],
          {ax, bx, cx, dx}][[i]].
       {av, bv, cv, dv}[[i]],
     {i, 4}];
```

[1]http://www.wolfram.com/mathematica/

The force, F_s, applied on each of the four cells of a tetrahedron can then be succinctly expressed as:

```
F[ax_,bx_,cx_,dx_,av_,bv_,cv_,dv_,rest_,skd_,skdamp_]:=
     -(skd*CTet[ax,bx,cx,dx,rest] +
     TetDamping[ax,bx,cx,dx,av,bv,cv,dv,rest ,skdamp])*
     Grad[CTet[ax,bx,cx,dx,rest],{ax,bx,cx,dx}];
```

Some final commands allow us to express the force as a function of the four cells' positions (a, b, c, d) and velocities (a', b', c', d'), the rest volume of the tetrahedron, R, its stiffness coefficient, s_{kd}, and damping coefficient, s_{kdamp}:

```
Table[ToExpression[letter <> "X = " <> ToString[InputForm[
        Table[
         ToExpression[
          "Subscript["<>ToLowerCase[letter]<>","<>index<>"]"],
         {index, {"x", "y", "z"}}]]]]],
  {letter, {"A", "B", "C", "D"}}];
Table[ToExpression[letter <> "V = " <> ToString[InputForm[
        Table[
         ToExpression[
          "Subscript["<>ToLowerCase[letter]<>"',"<>index<>"]"],
         {index, {"x", "y", "z"}}]]]]],
  {letter, {"A", "B", "C", "D"}}];
F[AX,BX,CX,DX,AV,BV,CV,DV,R,Subscript[s,kd],Subscript[s,kdamp]]
 // FullSimplify
```

Executing this last command in Mathematica gives the full expansion of F_s as a four-tuple, $\{F_a, F_b, F_c, F_d\}$, that contains the three dimensional forces acting on each of the cells. This equation can then be implemented in a procedural language as part of the physical integrator.

References

Agarwal, M. and Cagan, J. (1998). A blend of different tastes: the language of coffeemakers, *Environment and Planning B: Planning and Design* **25**: 205–226.

Agarwal, P. (1994). The cell programming language, *Artificial Life* **2**(1): 37–77.

Allen, M. T., Prusinkiewicz, P. and DeJong, T. M. (2005). Using L-systems for modeling sourcesink interactions, architecture and physiology of growing trees: the L-PEACH model, *New Phytologist* **166**(3): 869–880.

Ball, P. (2001). *The Self-Made Tapestry: Pattern Formation in Nature*, Oxford University Press, Oxford.

Baraff, D. and Witkin, A. (1992). Dynamic simulation of non-penetrating flexible bodies, *SIGGRAPH Computer Graphics* **26**(2): 303–308.

Baraff, D. and Witkin, A. (1997). Physically-based modeling, principles and practice, *Course Notes for SIGGRAPH'97*, ACM.

Barthe, L., Wyvill, B. and De Groot, E. (2004). Controllable binary CSG operators for "soft objects", *International Journal of Shape Modeling* **10**(2): 135–154.

Bentley, P. J. and Corne, D. W. (eds) (2002). *Creative Evolutionary Systems*, number 576, Academic Press, London.

Berlekamp, E. R., Conway, J. H. and Guy, R. K. (1982). *Winning Ways for your Mathematical Plays*, Vol. 2, Academic Press, New York.

Blinn, J. F. (1982). A generalization of algebraic surface drawing, *ACM Transactions on Graphics* **1**(3): 235–256.

Bloomenthal, J. (1985). Modeling the mighty maple, *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, ACM, New York, NY, USA, pp. 305–311.

Bloomenthal, J. (1995). *Skeletal design of natural forms*, PhD thesis, Calgary, Alberta.

Bloomenthal, J. and Bajaj, C. (1997). *Introduction to Implicit Surfaces*, Morgan Kaufmann, San Francisco, CA.

Bridson, R., Fedkiw, R. and Anderson, J. (2005). Robust treatment of collisions, contact and friction for cloth animation, *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, p. 2.

Brodland, G. W. (2004). Computational modeling of cell sorting, tissue engulfment, and related phenomena: a review, *Applied Mechanics Reviews* **57**(1): 47–76.

Catmull, E. and Clark, J. (1978). Recursively generated B-spline surfaces on arbitrary topological meshes, *Computer-Aided Design* **10**(6): 350 – 355.

Chau, H. H., Chen, X., McKay, A. and de Pennington, A. (2004). Evaluation of a 3D shape grammar implementation, *in* J. Gero (ed.), *Design Computing and Cognition '04*, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 357–376.

Cickovski, T. M., Huang, C., Chaturvedi, R., Glimm, T., Hentschel, H. G. E., Alber, M. S., Glazier, J. A., Newman, S. A. and Izaguirre, J. A. (2005). A framework for three-dimensional simulation of morphogenesis, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **2**(3): 273–288.

Combaz, J. and Neyret, F. (2002). Painting folds using expansion textures, *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, p. 176.

Combaz, J. and Neyret, F. (2006). Semi–interactive morphogenesis, *Proceedings of the IEEE International Conference on Shape Modeling and Applications*, Matsushima, Japan, p. 35.

Crassin, C., Neyret, F., Lefebvre, S. and Eisemann, E. (2009). Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering, *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, ACM Press, Boston, MA.

Cummings, F. W. (2001). The interaction of surface geometry with morphogens, *Journal of Theoretical Biology* **212**(3): 303–313.

Davies, J. A. (2005). *Mechanisms of morphogens: the creation of biological form*, Elsevier, Burlington, MA.

de Boer, M. J. M., Fracchia, F. D. and Prusinkiewicz, P. (1992). A model for cellular development in morphogenetic fields, *Lindenmayer Systems: Impacts on*

Theoretical Computer Science, Computer Graphics, and Developmental Biology, Springer-Verlag, pp. 351–370.

Dellaert, F. and Beer, R. D. (1994). Toward an evolvable model of development for autonomous agent synthesis, *in* P. Maes and R. Brooks (eds), *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, MA, pp. 246–257.

Drewes, F. and Kreowski, H.-J. (1997). Picture generation by collage grammars, *in* H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds), *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific Publishing Co., Inc., River Edge, NJ, pp. 397–457.

Dummer, J. (2005). A simple time-corrected verlet integration method. (retrieved on 29/12/2009).
URL: *http://www.gamedev.net/reference/articles/article2200.asp*

Duvdevani-Bar, S. and Segel, L. (1988). On topological simulations in developmental biology, *Journal of Theoretical Biology* **131**(1): 33–42.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K. and Worley, S. (2003). *Texturing & Modeling: A Procedural Approach*, third edn, Morgan Kaufmann, San Francisco, CA.

Eden, M. (1961). A two-dimensional growth process, *in* J. Neyman (ed.), *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume IV: Biology and Problems of Health*, The Regents of the University of California, pp. 223–239.

Eggenberger, P. (2003). Genome-physics interaction as a new concept to reduce the number of genetic parameters in artificial evolution, *in* R. Sarker, R. Reynolds, H. Abbass, K.-C. Tan, R. McKay, D. Essam and T. Gedeon (eds), *Proceedings of the IEEE 2003 Congress on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pp. 191–198.

Ermentrout, G. B. and Edelstein-Keshet, L. (1993). Cellular automata approaches to biological modeling, *Journal of Theoretical Biology* **160**(1): 97–133.

Federl, P. and Prusinkiewicz, P. (2004). Finite element model of fracture formation on growing surfaces, *in* M. Bubak, G. van Albada, P. Sloot and J. Dongarra (eds), *Proceedings of Computational Science. ICCS 2004* (Krakow, Poland, June 6–9, 2004), *Part II, Lecture Notes in Computer Science 3037*, Springer, Berlin, pp. 138–145.

Flake, G. W. (1999). *The Computational Beauty of Nature*, MIT Press, Cambridge, Massachusetts.

Fleischer, K. (1995). *A Multiple-Mechanism Developmental Model for Defining Self-Organizing Geometric Structures*, PhD thesis, California Institute of Technology, Pasadena, California.

Fleischer, K. W., Laidlaw, D. H., Currin, B. L. and Barr, A. H. (1995). Cellular texture generation, *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, ACM Press, New York, NY, pp. 239–248.

Foley, J. D., Dam, A. v., Feiner, S. K. and Hughes, J. F. (1990). *Computer graphics: principles and practice*, second edn, Addison-Wesley, Reading MA.

Forrest, A. (1980). The twisted cubic curve: a computer-aided geometric design approach, *Computer-Aided Design* **12**(4): 165 – 172.

Fournier, A., Bloomenthal, J., Oppenheimer, P., Reeves, W. T. and Smith, A. R. (1987). *The Modelling of Natural Phenomena*, Vol. 16 of *SIGGRAPH '87 Course Notes*, ACM SIGGRAPH, Anaheim, CA.

Giavitto, J. L., Godin, C., Michel, O. and Prusinkiewicz, P. (2002). Computational models for integrative and developmental biology, *Technical report*, Université d'Évry Val d'Essonne.

Gibson, S. F. (1997). 3D chainmail: a fast algorithm for deforming volumetric objects, *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, ACM, New York, NY, USA, pp. 149–154.

Gibson, S. F. and Mirtich, B. (1997). A survey of deformable modeling in computer graphics, *Technical report*, Mitsubishi Electric Research Laboratories.

Gilbert, S. F. (2006). *Developmental Biology*, 8th edn, Sinauer Associates, Inc., Sunderland, Massachusetts.

Greene, N. (1989). Voxel space automata: modeling with stochastic growth processes in voxel space, *SIGGRAPH '89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, pp. 175–184.

Grinspun, E., Hirani, A. N., Desbrun, M. and Schröder, P. (2003). Discrete shells, *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, pp. 62–67.

Haeckel, E. (1904). *Kunstformen der Natur.* (retrieved on 01/01/2011).
URL: *http://caliban.mpiz-koeln.mpg.de/~stueber/haeckel/kunstformen/natur.html*

Harrison, L. G., Wehner, S. and Holloway, D. M. (2001). Complex morphogenesis of surfaces: theory and experiment on coupling of reaction diffusion patterning to growth, *Faraday Discuss.* **120**: 277–294.

Heidelberger, B., Teschner, M., Keiser, R., Müller, M. and Gross, M. (2004). Consistent penetration depth estimation for deformable collision response, *Proceedings of Vision, Modeling, and Visualization*, Stanford, USA, pp. 339–346.

Heisserman, J. A. (1991). *Generative Geometric Design and Boundary Solid Grammars*, PhD thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Herman, G. T. and Rozenberg, G. (1975). *Developmental Systems and Languages*, North-Holland Publishing Company, Amsterdam, The Netherlands.

Hogeweg, P. (2000). Shapes in the shadow: Evolutionary dynamics of morphogenesis, *Artificial Life* **6**(1): 85–101.

Hogeweg, P. (2003). Evolving mechanisms of morphogenesis: on the interplay between differential adhesion and cell differentiation, *Journal of Theoretical Biology* **203**(4): 317–333.

Holton, M. (1994). Strands, gravity and botanical tree imagery, *Computer Graphics Forum* **13**(1): 57–67.

Honda, H. (1978). Description of cellular patterns by Dirichlet domains: The two–dimensional case, *Journal of Theoretical Biology* **72**(3): 523–543.

Irvine, D. H. (1988). Efficient solution of nonlinear models expressed in S-system canonical form, *Mathematical and Computer Modelling* **11**: 123 – 128.

Irving, G., Schroeder, C. and Fedkiw, R. (2007). Volume conserving finite element simulations of deformable models, *ACM Trans. Graph.* **26**(3).

Jirasek, C., Prusinkiewicz, P. and Moulia, B. (2000). Integrating biomechanics into developmental plant models expressed using L-systems, *Plant biomechanics 2000 . Proceedings of the 3rd Plant Biomechanics Conference, Freiburg-Badenweiler, August 27 to September 2, 2000.*, Georg Thieme Verlag, Stuttgart, pp. 615–624.

Joe, B. (1995). Construction of three-dimensional improved-quality triangulations using local transformations, *SIAM J. Sci. Comput.* **16**(6): 1292–1307.

Kaandorp, J. A. (1994). *Fractal Modelling: Growth and Form in Biology*, Springer-Verlag, Berlin, Germany.

Kaandorp, J. A. and Kübler, J. E. (2001). *The Algorithmic Beauty of Seaweeds, Sponges and Corals*, Springer-Verlag, Berlin, Germany.

Kauffman, S. (1995). *At home in the universe*, Oxford University Press, New York.

Kawaguchi, Y. (1996). The art of the growth algorithm, *in* C. G. Langton and K. Shimohara (eds), *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Nara, Japan, pp. 159–166.

Kharevych, L., Mullen, P., Owhadi, H. and Desbrun, M. (2009). Numerical coarsening of inhomogeneous elastic materials, *ACM Trans. Graph.* **28**(3): 51:1–51:8.

Kniemeyer, O., Buck-Sorlin, G. H. and Kurth, W. (2004). A graph grammar approach to artificial life, *Artificial Life* **10**(4): 413–431.

Krul, T., Kaandorp, J. A. and Blom, J. G. (2003). Modelling developmental regulatory networks, *Computational Science - ICCS 2003, Pt IV, Proceedings. Lecture Notes in Computer Science.*, Vol. 2660, Springer, pp. 688–697.

Lam, Z. and King, S. A. (2005). Simulating tree growth based on internal and environmental factors, *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM Press, New York, NY, USA, pp. 99–107.

Lantin, M. (1997). Computer simulations of developmental processes, *Technical report*, SFU CMPT. (retrieved on 29/12/2009).
URL: *ftp://fas.sfu.ca/pub/cs/TR/1997/CMPT97-24.pdf*

Leung, C. H. and Berzins, M. (2003). A computational model for organism growth based on surface mesh generation, *J. Comput. Phys.* **188**(1): 75–99.

Lin, S., Lee, Y.-S. and Narayan, R. J. (2010). Heterogeneous deformable modeling of bio-tissues and haptic force rendering for bio-object modeling, *in* R. Narayan, T. Boland and Y.-S. Lee (eds), *Printed Biomaterials*, Springer, pp. 19–37.

Lindenmayer, A. (1967). An axiom system for the development of filamentous organisms, *Abstracts of the III International Congress on Logic, Methodology and Philosophy of Science*, Amsterdam, pp. 127–128.

Lindenmayer, A. and Rozenberg, G. (1979). Parallel generation of maps, *Developmental systems for cell layers, Lecture Notes in Computer Science*, Vol. 73, Springer-Verlag, Berlin, pp. 301–316.

Lintermann, B. and Deussen, O. (1998). A modelling method and interface for creating plants, *Computer Graphics Forum* **17**(1): 73–82.

Loop, C. (1987). *Smooth subdivision surfaces based on triangles*, Master's thesis, University of Utah.

Maierhofer, S. (2002). *Rule-Based Mesh Growing and Generalized Subdivision Meshes*, PhD thesis, Vienna University of Technology.

Matela, R. J. and Fletterick, R. J. (1979). A topological exchange model for cell self-sorting, *Journal of Theoretical Biology* **76**(4): 403–414.

Matela, R. J. and Fletterick, R. J. (1980). Computer simulation of cellular self-sorting: A topological exchange model, *Journal of Theoretical Biology* **84**(4): 673–690.

Matela, R. J., Random, R. and Bowles, M. A. (1983). Computer simulation of compartment maintenance in the *Drosophila* wing imaginal disc, *Journal of Theoretical Biology* **103**(3): 357–378.

McCormack, J. (2005). A developmental model for generative media, *in* M. S. Capcarrere, A. A. Freitas, P. J. Bentley, C. G. Johnson and J. Timmis (eds), *Lecture Notes in Artificial Intelligence (Proceedings of the 8th European Conf. on Advances in Artificial Life)*, Vol. 3630, Springer–Verlag, pp. 88–97.

McCormack, J., Dorin, A. and Innocent, T. (2004). Generative design: a paradigm for design research, *in* J. Redmond, D. Durling and A. de Bono (eds), *Futureground*, Design Research Society, Melbourne.

Meinhardt, H. (1982). *Models of Biological Pattern Formation*, Academic Press, London.

Meinhardt, H. (2003). *The Algorithmic Beauty of Sea Shells*, Springer-Verlag, Berlin, Germany.

Mjolsness, E., Sharp, D. H. and Reinitz, J. (1991). A connectionist model of development, *Journal of Theoretical Biology* **152**(4): 429–453.

Müller, M., Heidelberger, B., Hennix, M. and Ratcliff, J. (2007). Position based dynamics, *J. Vis. Comun. Image Represent.* **18**(2): 109–118.

Müller, M., Heidelberger, B., Teschner, M. and Gross, M. (2005). Meshless deformations based on shape matching, *Proceedings of SIGGRAPH'05*, Los Angeles, CA, pp. 471–478.

Müller, M., Stam, J., James, D. and Thürey, N. (2008). Real time physics: class notes, *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, pp. 1–90.

Měch, R. and Prusinkiewicz, P. (1996). Visual models of plants interacting with their environment, *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, pp. 397–410.

Nealen, A., Mueller, M., Keiser, R., Boxerman, E. and Carlson, M. (2006). Physically Based Deformable Models in Computer Graphics, *Computer Graphics Forum* **25**(4): 809–836.

Nealen, A., Müller, M., Keiser, R., Boxermann, E. and Carlson, M. (2005). Physically Based Deformable Models in Computer Graphics (State of the Art Report), *Proceedings of Eurographics*, Dublin, Ireland, pp. 71–94.

O'Brien, J. F. and Hodgins, J. K. (1999). Graphical modeling and animation of brittle fracture, *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 137–146.

Popović, J. and Hoppe, H. (1997). Progressive simplicial complexes, *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 217–224.

Porter, B. (2009a). A developmental system for organic form synthesis, *Technical Report 2009/245*, Monash University, Clayton School of Information Technology. **URL:** *http://www.csse.monash.edu.au/publications/2009/tr-2009-245-full.pdf*

Porter, B. (2009b). A developmental system for organic form synthesis, *in* K. B. Korb, M. Randall and T. Hendtlass (eds), *ACAL*, Vol. 5865 of *Lecture Notes in Computer Science*, Springer, pp. 136–148.

Porter, B. and McCormack, J. (2010). Developmental modelling with SDS, *Computers & Graphics* **34**(4): 294 – 303.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*, third edn, Cambridge University Press, Cambridge, UK.

Prusinkiewicz, P. (1993). Modeling and visualization of biological structures, *Proceeding of Graphics Interface '93*, Toronto, Ontario, pp. 128–137.

Prusinkiewicz, P. (2004). Modeling plant growth and development, *Current Opinion in Plant Biology* **7**(1): 79–83.

Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*, Springer-Verlag, New York.

Prusinkiewicz, P. and Lindenmayer, A. (1996). *The Algorithmic Beauty of Plants*, 2nd edn, Springer-Verlag.

Ransom, R. and Matela, R. J. (1984). Computer modelling of cell division during development using a topological approach, *Journal of Embryology and Experimental Morphology* **83**: 233–259.

Reeves, W. T. (1983). Particle systems-a technique for modeling a class of fuzzy objects, *ACM Transactions on Graphics* **2**(2): 91–108.

Rozenberg, G. and Salomaa, A. (1980). *The Mathematical Theory of L-systems*, Academic Press, New York.

Rozenberg, G. and Salomaa, A. (1986). *The Book Of L*, Springer-Verlag, Berlin.

Sandberg, A. (2006). Models of development, *Technical report*, KTH, Stockholm. (retrieved on 20/06/2007).
URL: *http://www.nada.kth.se/~asa/Work/index.html*

Sanderson, A. R., Kirby, R. M., Johnson, C. R. and Yang, L. (2006). Advanced reaction-diffusion models for texture synthesis, *Journal of Graphics Tools* **11**(3): 47–71.

Schneider, P. J. and Eberly, D. H. (2003). *Geometric tools for computer graphics*, The Morgan Kaufmann series in computer graphics and geometric modeling, Morgan Kaufmann, San Francisco, CA.

Sick, S., Reinker, S., Timmer, J. and Schlake, T. (2006). WNT and DKK determine hair follicle spacing through a reaction-diffusion mechanism, *Science* **314**: 1447–1450.

Sifakis, E., Shinar, T., Irving, G. and Fedkiw, R. (2007). Hybrid simulation of deformable solids, *in* D. Metaxas and J. Popovic (eds), *ACM SIGGRAPH Symposium on Computer Animation*, San Diego, CA, pp. 81–90.

Sims, K. (1994). Evolving virtual creatures, *Computer Graphics*, ACM SIGGRAPH, pp. 15–22.

Smith, A. R. (1984). Plants, fractals, and formal languages, *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, ACM, New York, NY, USA, pp. 1–10.

Smith, C. (2006). *On Vertex-Vertex Systems and their use in geometric and biological modelling*, PhD thesis, The University of Calgary.

Stanley, K. O. (2006). Exploiting regularity without development, *Proceedings of the AAAI Fall Symposium on Developmental Systems*, AAAI Press, Menlo Park, CA.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny, *Artificial Life* **9**(2): 93–130.

Stiny, G. (1977). Ice-ray: A note on the generation of chinese lattice designs, *Environment and Planning B* **4**(1): 89–98.

Stiny, G. and Gips, J. (1972). Shape grammars and the generative specification of painting and sculpture, *in* C. V. Friedman (ed.), *Information Processing '71*, Amsterdam, pp. 1460–1465.

Stiny, G. and Gips, J. (1978). *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts*, University of California Press, Berkeley; Los Angeles, CA.

Streichert, F., Spieth, C., Ulmer, H. and Zell, A. (2003). Evolving the ability of limited growth and self-repair for artificial embryos, *Proceedings of the 7th European Conference on Artificial Life*, Springer-Verlag, pp. 289–298.

Terzopoulos, D. and Fleischer, K. (1988). Deformable models, *The Visual Computer* **4**: 306–331.

Teschner, M., Heidelberger, B., Mueller, M., Pomeranets, D. and Gross, M. (2003). Optimized spatial hashing for collision detection of deformable objects, *Proceedings of Vision, Modeling, and Visualization*, Munich, Germany, pp. 47–54.

Teschner, M., Heidelberger, B., Müller, M. and Gross, M. (2004). A versatile and robust model for geometrically complex deformable solids, *Proceedings of Computer Graphics International*, Heraklion, Greece, pp. 312–319.

Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N. and P.Volino, W. S. (2005). Collision detection for deformable objects, *Computer Graphics Forum* **24**(1): 61–81.

Thom, R. (1975). *Structural stability and morphogenesis: an outline of a general theory of models*, 1st ed. english edn, Benjamin, Reading, Mass.

Thompson, D. W. (1942). *On Growth and Form*, second edn, Cambridge University Press, Cambridge, UK.

Tonnesen, D. (1992). Spatially coupled particle systems, *SIGGRAPH'92 Course Notes #16: Particle System Modeling, Animation, and Physically Based Techniques*, ACM, New York, NY.

Turing, A. M. (1952). The chemical basis of morphogenesis, *Philosophical Transactions of the Royal Society* **237**(641): 37–72.

Turini, G., Pietroni, N., Ganovelli, F. and Scopigno, R. (2007). Techniques for computer assisted surgery, *Eurographics Italian Chapter Conference 2007*.

Turk, G. (1991). Generating textures on arbitrary surfaces using reaction-diffusion, *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, pp. 289–298.

Ulam, S. (1962). On some mathematical properties connected with patterns of growth on figures, *Proceedings of Symposia on Applied Mathematics*, Vol. 14, American Mathematical Society, pp. 215–224.

Vaario, J. (1994). From evolutionary computation to computational evolution, *Informatica* **18**(4): 417–434.

Von Neumann, J. (1966). *Theory of Self-Reproducing Automata*, University of Illinois Press.

Weliky, M. and Oster, G. (1990). The mechanical basis of cell rearrangement: I. Epithelial morphogenesis during *Fundulus* epiboly, *Development* **109**(2): 373–386.

Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R. and O'Brien, J. F. (2010). Dynamic local remeshing for elastoplastic simulation, *ACM Trans. Graph.* **29**(4): 49:1–49:11.

Witten, T. A. and Sander, L. M. (1981). Diffusion-limited aggregation, a kinetic critical phenomenon, *Phys. Rev. Letters* **47**(19): 1400–1403.

Wolpert, L. (1969). Positional information and the spatial pattern of cellular differentiation, *Journal of Theoretical Biology* **25**(1): 1–47.

Zorin, D. and Schröder, P. (2000). Subdivision for modeling and animation, *Course Notes for SIGGRAPH 2000*, ACM, New York, NY.