Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics

by

Radhika Nagpal

S.B., Massachusetts Institute of Technology (1994)

S.M., Massachusetts Institute of Technology (1994)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

Author ....

Department of Electrical Engineering and Computer Science

May 25, 2001

Certified by ....

Gerald Jay Sussman

Matsushita Professor of Electrical Engineering, MIT

Thesis Supervisor

Certified by ....

Harold Abelson

Class of 1922 Professor of Computer Science and Engineering, MIT

Thesis Supervisor

Accepted by ....

Arthur C. Smith

Chairman, Department Committee on Graduate Students

Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics

by

Radhika Nagpal

Submitted to the Department of Electrical Engineering and Computer Science on May 25, 2001, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

In this thesis I present a language for instructing a sheet of identically-programmed, flexible, autonomous agents ("cells") to assemble themselves into a predetermined global shape, using local interactions. The global shape is described as a folding construction on a continuous sheet, using a set of axioms from paper-folding (origami). I provide a means of automatically deriving the cell program, executed by all cells, from the global shape description.

With this language, a wide variety of global shapes and patterns can be synthesized, using only local interactions between identically-programmed cells. Examples include flat layered shapes, all plane Euclidean constructions, and a variety of tessellation patterns. In contrast to approaches based on cellular automata or evolution, the cell program is directly derived from the global shape description and is composed from a small number of biologically-inspired primitives: gradients, neighborhood query, polarity inversion, cell-to-cell contact and flexible folding. The cell programs are robust, without relying on regular cell placement, global coordinates, or synchronous operation and can tolerate a small amount of random cell death. I show that an average cell neighborhood of 15 is sufficient to reliably self-assemble complex shapes and geometric patterns on randomly distributed cells.

The language provides many insights into the relationship between local and global descriptions of behavior, such as the advantage of constructive languages, mechanisms for achieving global robustness, and mechanisms for achieving scale-independent shapes from a single cell program. The language suggests a mechanism by which many related shapes can be created by the same cell program, in the manner of D'Arcy Thompson's famous coordinate transformations. The thesis illuminates how complex morphology and pattern can emerge from local interactions, and how one can engineer robust self-assembly.

Thesis Supervisor: Gerald Jay Sussman
Title: Matsushita Professor of Electrical Engineering, MIT

Thesis Supervisor: Harold Abelson
Title: Class of 1922 Professor of Computer Science and Engineering, MIT

Acknowledgments

Credit for this thesis is shared by my family: my parents, Dr Vijay Nagpal and Kamini Nagpal, my husband Quinton Y. Zondervan, my brother Gaurav Nagpal and my daughter Jahnavi Joyce Zondervan. Their love and support made this possible.

Gerry Sussman, Hal Abelson and Tom Knight, for fostering an environment for novel ideas and non-traditional thinking. Over the years they have not only shared many wild and crazy ideas, but also many philosophies of research and research culture that I am now beginning to appreciate. I especially thank Gerry for his untiring encouragement and enthusiasm during this last year; it has been fun.

Alan Berenbaum, my mentor of many years and the Bell Labs GRPW Fellowship Committee who provided funding for all of my graduate career with no strings attached, no questions asked, and mentoring for free. For me, Bell labs will always represent a vision of freedom in research.

The work in this thesis stems from many years of collaboration and discussions with people in and around the Amorphous Computing Group: Daniel Coore, Ron Weiss, Kevin Lin, Chris Hanson, Stephen Adams, Eric Rauch, Jacob Katznelson, Norm Margolus, Raissa D'Souza, Chris Lass, Piotr Mitros, Jake Beal, Bill Butera, Geo Homsy, and Rebecca Frankel.

Professor Tom Hull of Merrimack College, who not only provided me with the origami mathematics papers that are otherwise difficult to find, and answered my million and one questions, but also introduced me to the small community of mathematicians whose work has influenced this thesis.

Chris Hanson, for his help in every possible aspect of completing this dissertation: tweaking numerical simulations, to installing debian on my laptop, to advice on child-rearing. I have always admired his amazing skills, but I have discovered that his most valuable skill is the generosity with which he shares the rest.

Becky Bisbee and Jessica Strong, for making road blocks suddenly dissappear.

Holly Yanco, for reading the thesis, many lunches, and the wheelchair ride.

Sunil Vemuri, for his sincere friendship, distractions and babysitting.

My friends, people at Bell Labs, past members of the Programming Methodology group, and all those people who have shared part of the journey.

Contents

List of Figures

Chapter 1

Introduction

It is fascinating how global phenomena can emerge from the local interactions of millions of individuals, with simple rules and no global intent or understanding. The individuals may be simple and unreliable, but the end result can be very complex and robust. Colonies of ants and termites cooperate to achieve global tasks and build complex structures that can not be achieved by an individual [28]. Even more astounding is the precision and reliability with which cells with identical DNA cooperate to form complex structures, such as ourselves, starting from a nearly homogeneous egg [42]. Such examples provide an important inspiration for engineering systems with vast numbers of parts — we would like to achieve the kind of robustness and complexity that biology achieves.

At the same time it is becoming critically important to understand how to design and program large decentralized systems. Technology like MEMs (micro-electro-mechanical devices) is making it possible to bulk-manufacture millions of tiny computing elements integrated with sensors and actuators and embed these into materials, structures and the environment. Already many novel applications are being envisioned: computing elements that can be "painted" onto a surface, sensor-covered beams that actively resist buckling, an airplane wing that changes shape to resist shear, a programmable assembly line of air valves, a reconfigurable robot that morphs into different shapes to achieve different tasks. [10, 69, 8, 18, 11, 74, 73]. Emerging research in biocomputing may even make it possible to harness the many sensors and actuators in cells and create programmable tissues [34, 68, 67]. Widespread application of these technologies is imminent, yet programming them still remains a challenge.

This thesis provides an example of how complex morphology and pattern can emerge from local interactions, and how one can engineer robust self-assembling systems. The inspiration for my work comes from developmental biology. The goal is to address the following challenges: a) How does one design local behavior to achieve a *particular* global behavior b) What are the appropriate local and global paradigms for engineering such systems?

1.1 Thesis Statement

In this thesis I present a language for instructing a sheet of identically-programmed, flexible, autonomous agents ("cells") to assemble themselves into a predetermined global shape, using local interactions. The desired global shape is described as a folding construction on a continuous sheet, using a set of axioms from paper-folding (origami). I provide a means of automatically deriving the cell program, executed by all cells, from the global shape description. The cell program is inspired by developmental biology. With this language, a wide variety of global shapes and patterns can be described at an abstract level, and then synthesized using only local interactions between identically-programmed cells.

1.2 Approach

The programmable cell sheet is inspired by epithelial cell tissues, that form a variety of structures through the coordination of local shape changes in individual epithelial cells. The programmable cell sheet consists of a single layer of identically-programmed cells that can coordinate to fold the sheet along straight lines. The cells are autonomous and execute programs based on purely local interactions with nearby cells. Cells can also communicate with other cells that come into direct contact as a result of changes in the shape of sheet. Each cell has limited resources and reliability. The cells are randomly and densely distributed in the sheet; there is no global coordinate system, no global clock, and no centralized control.

The self-assembly works as follows: The global shape is described as a folding construction on a continuous sheet, using a language I developed called the Origami Shape Language. The language is inspired by origami, where a variety of complex shapes can be constructed from a sheet by using a small set of axioms, simple initial conditions (edges and corners of the sheet) and only two types of folds. The program for an individual cell is automatically compiled from the global shape description. All cells execute the same program and differ only in a small amount of local dynamic state. The cell program itself is inspired by biology and is composed from a small set of primitives: gradients, neighborhood query, cell-to-cell contact, polarity inversion and flexible folding. When the cell program is executed by all the cells in the sheet, the sheet is configured into the desired shape (figure 1-1).

1.3 Contributions

- With the Origami Shape Language, a wide variety of global shapes and patterns can be synthesized. Examples include flat layered shapes, all plane Euclidean constructions, and a variety of tessellation patterns. All of these shapes are achieved using cell programs based on purely local communication between cells and local sensing and actuation by cells. All cells execute the same program and differ only in a small amount of dynamic local state.

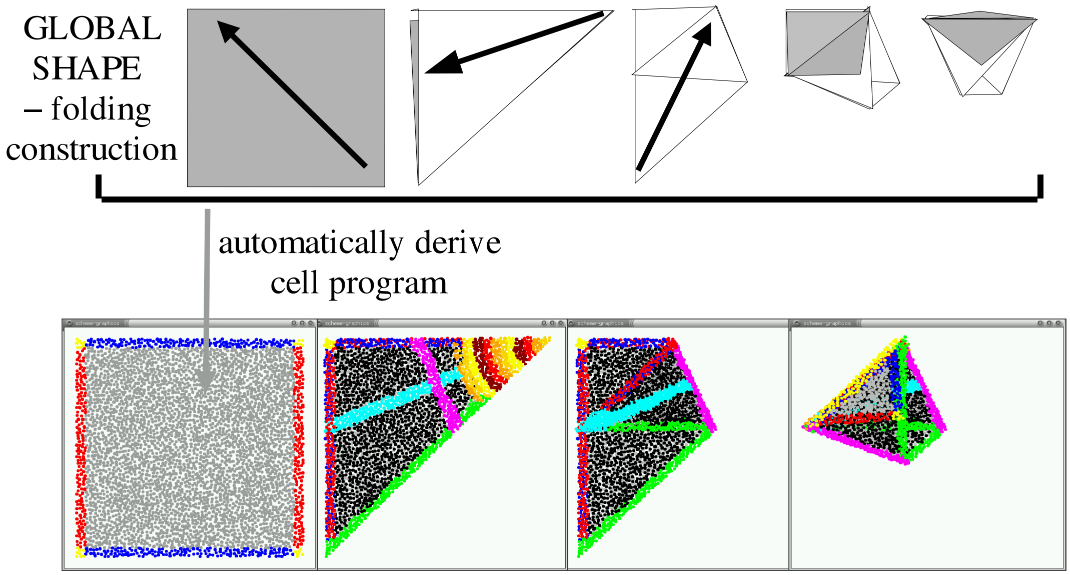


Figure 1-1: Overview of self-assembly approach: The global shape is described as a folding construction on a continuous sheet, and is specified using the Origami Shape Language. The program for an individual cell is automatically derived from the global shape program., using biologically-inspired primitives. The cell program is executed by identically-programmed, locally-interacting, autonomous cells in a sheet. The cells coordinate to fold the sheet into the desired shape.

- The cell program is directly compiled from the global shape description. This differs significantly from approaches based on cellular automata where local rules are constructed empirically or evolutionary approaches where the local-to-global relationship is not well-understood. By contrast I provide a small set of primitives for organization at the local level. These primitives are inspired by biologists' understanding of how pattern and morphology appear in the development of embryos such as the *Drosophila* and sea urchin [55, 42, 56]. These primitives are composed in predictable ways to create the cell program for a given shape. Like any other emergent system, the eventual shape emerges as a result of the local interactions between the elements. The compilation process confers many advantages; for example, the global shape can be described at an abstract level and results from origami mathematics can be used to reason about what kinds of shapes can and cannot be self-assembled by the cells.

- The cell program are robust, without relying on regular cell placement, global coordinates, or synchronous operation and can tolerate a small amount of random cell death. Instead, robustness is achieved by depending on large and dense cell populations, using average behavior rather than individual behavior, trading off precision for reliability, and avoiding any centralized control. There are no global coordinates, instead cells "discover" positional information as and when needed, using primitives inspired by biological systems. Through a combination of theoretical and experimental analysis, I show that an average local neighborhood of 15–20 cells is sufficient to reliably control the self-assembly of shapes and geometric patterns on randomly distributed cells. These properties are extremely attractive from an engineering perspective and the concepts are likely to have general applicability to programming large decentralized systems.

- The language provides many insights into the relationship between local and global descriptions of behavior. For example, the cell program is scale independent which implies it can create the same shape at many different scales without modification, simply by increasing the size of the cell sheet. I show that many related shapes can be created by the same cell program, by modifying the initial conditions of the sheet — which suggests a mechanism for achieving shape transformations in the manner of D'Arcy Thompson's famous examples [64]. The language exhibits several other suggestive biological properties. These properties provide insights into how complex morphology and pattern can emerge from cell interactions and in chapter 7 I discuss how these ideas can be used to direct biological experiments.

1.4 Context

There have been a variety of approaches to try and understand how global phenomena emerges from local interactions. Cellular automata and artificial life research use a bottom up, empirical approach: local rules are designed and the resulting systems are simulated to discover the emergent global dynamics. This research has mostly focused on studying natural systems: chemical pattern formation, thermodynamic self-assembly, traffic jams, and ant social behavior [43, 58, 17, 16, 62]. However this trial-and-error approach is difficult to use as an engineering tool. No framework is provided for constructing local rules to obtain any desired pattern; patterns instead "emerge" in a non-obvious way from the local interactions. One example is the application of ant-like behaviors to collections of mobile robots [16, 7, 47]. Interactions between these local rules can be quite complex and for the most part complex behavior is generated by using evolutionary or learning approaches [47]. Evolutionary approaches represent the opposite side of the spectrum. The approach is top down: the local rules are derived from the final global goal, but without any understanding of how or why they work. Not only is it difficult to verify the correctness of such local rules, constructing an appropriate fitness function can be as hard as designing a control algorithm from scratch [19, 49, 48].

As a result of the difficulty with these approaches, programming strategies within the MEMs community (where much of the underlying device technology is being built) have for the most part been centralized applications of traditional control theory. The few decentralized approaches assume access to global knowledge of the system and tend to focus on hierarchical control [8, 23, 69, 60]. Such centralized hierarchies are not scalable and can be quite brittle, catastrophically failing if a high-level node fails.

In the modular reconfigurable robotics community, most of the work has focused on centralized and heuristic searches, which quickly become intractable for large numbers of modules and often require untenable assumptions such as explicitly defining the final configuration and access to global position [57, 74, 21]. These programming strategies put further pressure to build complex, precise (and thus expensive) elements and interconnects rather than cheap, unreliable, mass-produced computing elements that one can conceive of just throwing at a problem. In the application environments envisioned, vast numbers of smart sensors and actuators are embedded into materials. It is unrealistic to control each element individually or rely on perfectly regular grids of perfectly reliable elements. The elements are likely to have limited power and resources, and only be able to gather information locally through sensing and communication. Depending on centralized information, like global clocks or external beacons for triangulating position, puts severe limitations on the types of applications and environments, and exposes easily attackable points of failures. Most importantly, it is not clear that these kinds of assumptions are necessary in order to achieve robust behavior.

Are there primitives for local organization that can be used to construct complex and reliable global behavior? Developmental biology suggests that there are general principles in place: the large similarities in DNA among all living things, the appearance of similar structures at wide varieties of scale, and simple mutations adding complete new structures such as wings or legs. The precision and reliability of most developmental processes, in the face of unreliable cells, variations in cell numbers and changes in the environment, is enough to make any engineer green with envy. In recent decades, there has been significant progress in understanding how cells produce complex pattern and shape during development [63, 42, 72, 6].

1.5 Related Work

My work is related to, and influenced by, the cellular automata and artificial life research but is more in line with the goal of Amorphous Computing [1], which is to identify engineering principles for systems that organize themselves to achieve predetermined global goals. The amorphous computing model is related to models such as cellular automata and StarLogo, but eschews regular grids and synchronous clocking in favor of a more amorphous setting: randomly distributed, asynchronous and possibly unreliable elements. The focus is on achieving robustness without assuming regularity and synchronization.

Many of the underlying primitives that I use have been developed collaboratively over the past five years in the amorphous computing group and are strongly influenced by ongoing research in developmental biology [54, 53, 68, 14]. In chapter 5 I discuss many interesting parallels between the Origami Shape Language and the Growing Point Language (GPL) developed by Daniel Coore in his PhD thesis [13]. Coore combined many amorphous computing primitives into a language for instructing cells to form any prespecified planar interconnect pattern. At the global level both the Origami Shape Language and GPL use constructive languages, but at the local level they use different strategies. As a result GPL produces patterns that guarantee topology whereas OSL produces faithful geometric patterns. OSL goes a step further by incorporating sensing and actuation in order to form shapes. Recently, work on a reconfigurable robot called Proteo has applied similar ideas to Coore's branching work to self-assemble branching and locomotion structures from mobile units [27]. The primitives used are very similar to those developed in amorphous computing, giving exciting evidence that such primitives have general applicability. One significant difference between OSL and previous work is that there is an abstraction barrier between the global and local descriptions of behavior — at the global level, the desired goal is specified without any notion of cells or modules.

My work shares a similar goal to other research on engineering self-assembling systems. This is a recent area of study with two main approaches: thermodynamic self-assembly and reconfigurable robots. In thermodynamic self-assembly the goal is to design materials with properties that cause them to assemble into the desired structure when mixed together. The work takes inspiration from molecular and chemical self-assembly [35, 25, 17]. Reconfigurable robotics is aimed at designing robots that can reconfigure themselves to suit different tasks. A robot consists of many mobile modules that can connect to form different shapes [21, 57, 73]. Proteo, for example, consists of small rhombic dodecahedron shaped units that always remain connected but can roll around each other to change the overall shape [74]. Currently all of the work in this area focuses on the formation of shapes from mobile units. My work explores a different metaphor — the formation of global shape through the coordination of local shape changes in connected units. What metaphors are appropriate for constructing different types of global shapes? All three approaches will play complementary roles towards finding an answer. In fact, biology suggests many more metaphors for creating complex structure such as growth, deposition of scaffolds, and cell death.

1.6 Outline

The next three chapters explain how the self-assembly works. Chapter 2 presents the programmable sheet model. Chapter 3 presents the Origami Shape Language. Chapter 4 presents the cell primitives and the compilation process by which the global goal is transformed into a cell program. Chapter 5 explores the different types of shapes and patterns that can be constructed. Chapter 6 provides a detailed analysis of the robustness of the system. Chapter 7 presents many interesting global properties of the language, such as scale-independence, and discusses analogies with biology.

Chapter 2

A Programmable Material

Imagine a flexible substrate, consisting of millions of tiny interwoven programmable fibers, that can be programmed to assume different shapes. Not only could one design many complex static structures from a single substrate, but also dynamic structures that react to, and affect, the environment. For example a programmable assembly line that moves objects by producing ripples in specific directions; manufacturing by programming; structures for deploying in space that fold compactly for storage but then unfold to perform the required function; a reconfigurable sheet that can change itself into a chair or a shelter or any other structure as needed. Programmable materials would make a host of novel applications possible that blur the boundary between computation and the environment.

Morphogenesis (creation of form) in developmental biology can provide insights for creating programmable materials that can change shape [6, 64, 63, 72]. Even basic developmental processes, such as gastrulation, are based on deliberate cell migration and shape change. Modeling such behaviors has been attracting increased attention from biologists [34, 36, 56]. Epithelial cells in particular generate a wide variety of structures: skin, capillaries, and many embryonic structures (gut, neural tube), through the coordinated effect of many cells changing their individual shape.

Our model for a programmable material is inspired by epithelial cell tissues — the programmable material is a sheet composed of identically programmed, connected, flexible cells that can fold the sheet through the coordination of local shape changes in individual cells. This is different from most reconfigurable robotics research that concentrates on substrates composed of mobile units that move around and connect in different ways to change the shape of the robot [21, 74, 57]. The programmable material uses a different metaphor, that of a flexible sheet. It is very versatile in the type of shapes and patterns that can be created. Biology suggests many other metaphors: growth, scaffolding, cell death, etc.

This chapter discusses the biological and mechanical underpinnings for the programmable sheet model as well as the computational model for an individual cell. This chapter also presents the simulation environment used in this thesis.

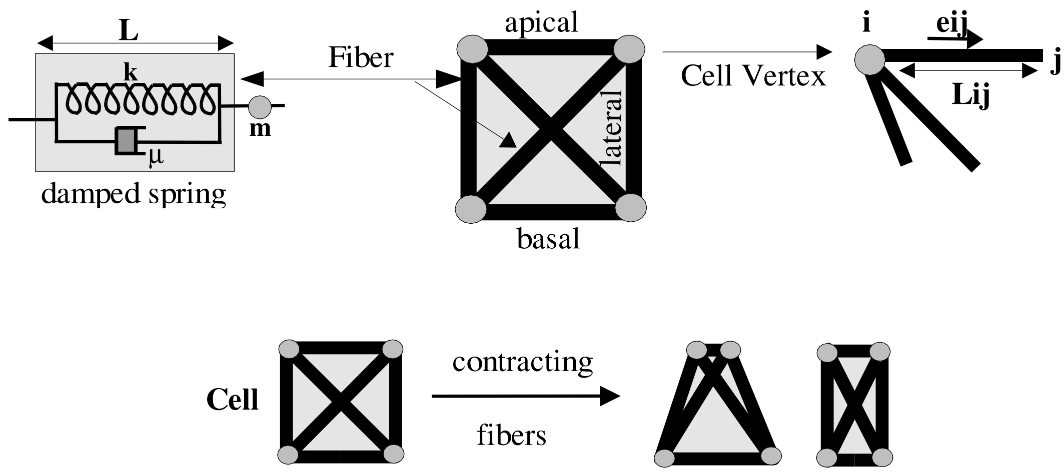


Figure 2-1: The model for a flexible cell, adapted from Odell *et al.* [56]. A cell has fibers along the edges and diagonals. A fiber is modeled as a damped spring. A cell can change shape by contracting its apical or basal fibers.

2.1 A Cell Model Inspired by Epithelial Cells

A seminal paper on mechanical models for morphogenesis is the paper on epithelial cell folding and invaginations by Odell, Oster, Alberch, and Burnside [56]. They proposed a mechanical model to explain how embryos form structures like the gut and neural tube through local deformations of individual epithelial cells. An epithelial cell has many fibers in its membranes. By contracting the fibers in its apical surface the cell can change it own shape; this is often referred to as *purse string contraction*. When many cells together use purse string contraction, invaginations can be formed. Odell *et al.* provided a mechanical model for an epithelial cell and presented many numerical simulations of rings of cells producing invaginations similar to those seen in the early development of sea urchin embryos.

The key insight is that the basis of the structure is a *simple flexible cell* that can actively deform its own shape and sense deformations in its shape. The global shape change is a result of many individual cells making local changes. This cell forms the basic element of a general programmable material.

Our cell model is similar to the mechanical model proposed by Odell *et al.* for the epithelial cell. In two dimensions the cell is modeled as a rectangle with six fibers: one apical (top), one basal (bottom), two lateral and two diagonal, with equal mass at each vertex as shown in figure 2-1. The cell has polarity, i.e. it can distinguish between the apical and basal sides.[1] The cell changes shape by contracting some set of its fibers. A fiber is modeled as a stiff, damped, rigid spring.

[1] Odell's model assumes that the cell is filled with an incompressible fluid so that it maintains volume, however for our purposes this part of the model is unnecessary and is not used when modeling 3D cells.

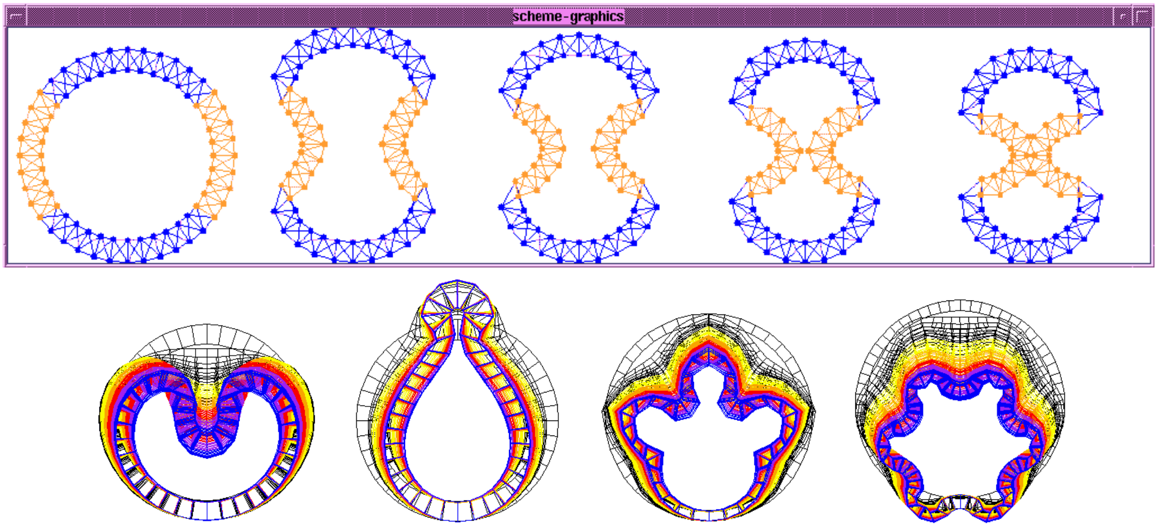


Figure 2-2: Different shapes can be formed by a ring of cells. In the upper simulation sequence, the orange cells are actively contracting. In the lower simulation pictures the different colors show how the shape changes over time.

$$m\frac{d^2L}{dt^2} = -k(L - L_0) - \mu\frac{dL}{dt} \tag{2.1}$$

Equation 2.1 models the dynamics of a single fiber, where m is mass, L is the length of the fiber, L_0 is the rest length of the spring, k is the spring stiffness constant, and μ is the damping constant. In equilibrium the length of the fiber depends on k and L_0 while μ affects the dynamic behavior. Contraction of the fiber is achieved by changing the stiffness k and rest length L_0 of the fiber.

The cell is modeled by a set of differential equations, one for each vertex. At any vertex of the cell, several fibers (from the same cell and possibly neighboring cells) apply force. The motion of a vertex under these forces is described by the equation:

$$m_i\frac{d^2p_i}{dt^2} = -\sum_{j \in adjvertices(i)} [k_{ij}(L_{ij} - L_{0ij}) + \mu_{ij}\frac{dL_{ij}}{dt}]e_{ij} \tag{2.2}$$

p_i is the position of vertex i, the subscript j refers to adjacent vertices. The subscript ij refers to the fiber between the vertices i and j and e_{ij} is the unit direction along fiber ij.

A substrate is formed by connecting cells along their lateral surfaces. The dynamics of the substrate is simulated by numerically integrating the set of differential equations for each vertex. One can simulate the effect of individual cells contracting their fibers. Odell *et al.* used a ring of cells filled with an incompressible fluid to model

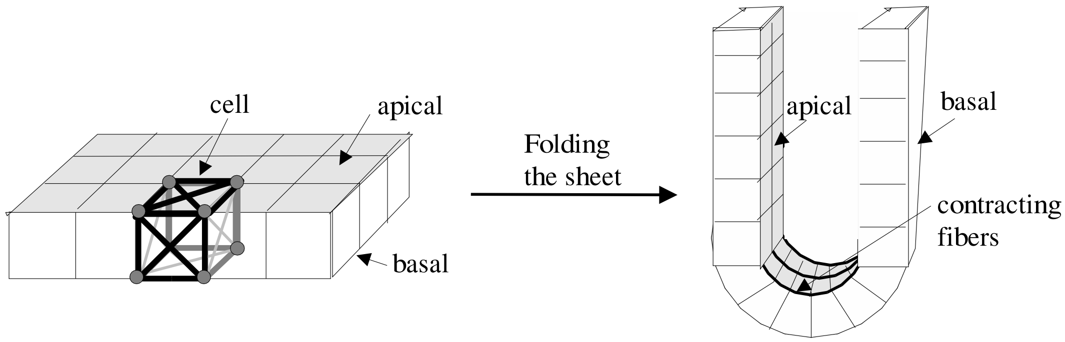


Figure 2-3: A sheet of flexible cells. The sheet can be folded by a line of cells contracting their apical or basal fibers that lie perpendicular to the direction of the desired fold line.

the formation of the gut and neural tube in embryos. Using the same ring model I simulated the formation of many other shapes, as shown in figure 2-2. In each case, individual cells were programmed to contract particular fibers. A nice feature of this model is that the shapes are robust to small changes in the ring parameters and fiber characteristics [56].

The cells can be connected to form many different substrates - tubes, solid blobs, etc. A sheet of cells is a very versatile substrate for forming different shapes. Figure 2-3 shows the model for a single-layered sheet of cells.[2] The cells all have the same apical-basal polarity. Figure 2-4 shows a variety of shapes that can be created from this sheet. The figures were generated by numerically simulating the differential equations for each vertex. The final results are shown using a VRML browser [12]. Most of the shapes shown are created by folding the sheet. The sheet can be folded by many cells contracting their apical or basal fibers that lie perpendicular to the direction of the desired fold line. The angle and curvature of the fold are determined by the extent of contraction and number of cells along the width of the line. In fact a wide variety of shapes can be produced by folding a sheet. One could also simulate dynamic shapes such as compression waves or ripples moving through the sheet.

2.2 A Programmable Cell Sheet

Ideally, we would like to experiment with creating much more complex shapes with much larger numbers of cells and non-regular cell placements. However the complexity of the numerical simulations quickly becomes intractable as the number of cells increase and the equations become stiffer as the number of active cells increase. If we do not increase the number of cells, the resolution is too limited to create complex shapes. Furthermore, one can not go far before self-intersection becomes a problem

[2] I extended Odell's 2D cell model to 3D with fibers on each edge and face diagonals (no body diagonals), but without the internal fluid.

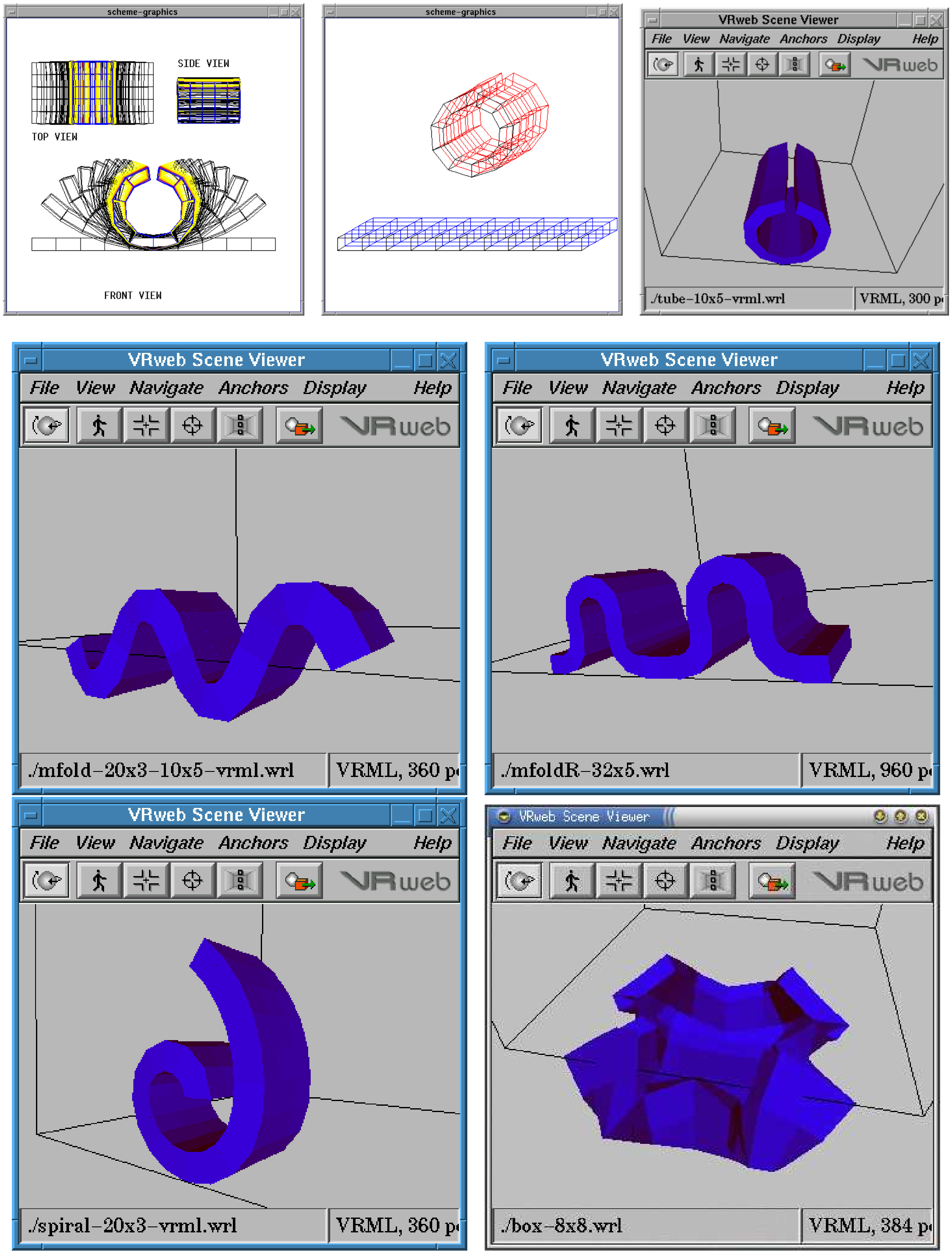


Figure 2-4: Different shapes formed by a sheet of cells.

and modeling that for even small numbers of cells can be a computational nightmare ([70, 5]). The stiff equations and self-intersection problems impose severe constraints on what can be simulated.

To bypass this problem, I use an abstract model of the sheet aimed at exploring shapes that can be created through folding. I focus on essentially one mechanical operation — cells coordinating to fold the sheet flat along a straight line. This type of cell sheet can be simulated without simulating the fibers.

The model for the programmable sheet used in this thesis, is a sheet consisting of thousands of randomly and densely distributed cells. The cells fill the space of the sheet (figure 2-5). The sheet has two surfaces: an apical surface and a basal surface. Each cell has an apical-basal polarity and can tell the difference between the two surfaces. If a line of cells all (virtually) fold their apical surfaces, then this causes the sheet to fold until its apical surface touches itself, and if all the cells fold their basal surfaces, then the sheet folds until the basal surface touches itself. The line of cells must have relatively uniform width and the cells must also be able to, on average, locally predict the direction of the fold. These requirements are based on how folding occurs in the dynamic model of the sheet. One reason for using flat folds is that there is currently no mechanism in the dynamic or abstract sheet model for individual cells to sense what the global angle of the fold is. Therefore folding precise angles is unrealistic. There are many interesting functional shapes that fold flat and in chapter 5 I present several examples as well as discuss extensions to self-stabilizing 3D shapes. The effect of straight flat folds on the configuration of the sheet is simple to simulate, and thus bypasses the limitations imposed by the numerical simulation of the dynamics. The simulation environment is discussed in section 2.4.

This abstract model allows us to simulate a sheet with thousands of randomly and densely distributed cells. With this resolution we can create many interesting shapes, without relying on regular grids, and guarantee robustness in the presence of failures. The next chapter discusses a way of thinking about creating shapes from a sheet using straight flat folds.

2.3 An Autonomous Cell

This section presents the computational model for the cell, which is based on the amorphous computing model [1]. All cells have *identical* programs, but an individual cell executes this program autonomously based on its communication with a small local neighborhood of cells. The total number of cells may be huge, but an individual cell can only communicate with an average of 15-20 nearby cells that are within a distance of r, using a local broadcast (figure 2-5).[3] Aside from a few simple initial conditions, cells have no knowledge of global position or interconnect topology. Nor are there any global observers; there is no global clock nor external beacons for

[3] The choice of 15-20 as the average neighborhood is explained in chapter 6. The idea of using a local broadcast is originally inspired by packet radio communication but also by in-plane capacitive communication.

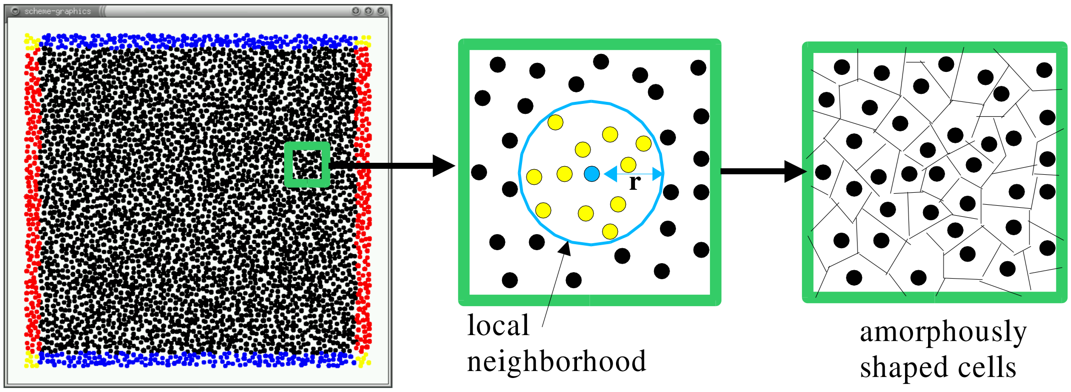


Figure 2-5: A programmable sheet of cells. Each cell communicates with only a small local neighborhood of cells within the communication radius r. The cells are randomly and densely distributed and assumed to fill the space of the sheet.

triangulating global position. A cell has limited memory for both code and local state, and also may die unexpectedly. The cells do not have unique global identities; rather, they have random-number generators that can be used to generate an identifier that has high probability of being locally unique (I am not concerned with global uniqueness). Cells have very simple sensing and actuator control; a cell can sense when another cell is in direct contact with its apical or basal surface, and a cell can virtually fold its apical or basal surface along a locally-determined orientation. A cell has limited ability to determine the local orientation between two neighbors in order to predict the fold orientation.[4]

The motivation for this cell model comes from the applications — we would like to cheaply bulk manufacture billions of smart sensors and actuators and embed them into materials and the environment. Assumptions such as globally unique identifiers, global clocks, global coordinates or perfectly reliable elements are unrealistic in this setting — especially if cells can discover these properties on their own. It is also necessary to avoid assumptions of precise interconnects and regular grids if these elements are to be embedded into materials and sprinkled onto surfaces. Furthermore, developmental biology suggests that it should be possible to construct complex structures without such assumptions. These considerations are a central piece of the amorphous computing paradigm and we believe that they will be necessary in order to make the vision of intelligent materials a reality.

[4]Unlike cellular automata, a cell does not know which neighbors are spatially opposite one another.

2.4 The Simulation Environment

The simulation environment used is Hlsim (High Level Simulator), which was developed as a testbed for amorphous computing ideas by Stephen Adams [3]. Hlsim simulates the execution of a program on a large number of asynchronous, identically-programmed, statically-placed elements with local communication. The program can be written in regular Scheme [2, 24], without worrying about the details of parallel execution. Hlsim converts the Scheme program into continuation-passing style and transparently executes and context switches between the elements. This not only alleviates the burden of thinking about parallel execution but also eliminates the possibility of fine-grain control over element execution order. Each element has its own local clock and receives and transmits messages within the local communication radius r. I wrote an extension to Hlsim to model how the sheet folds as a result of cell actuation.

Building a Cell Sheet: Figure 2-5 shows a simulation of a cell sheet with 1000 cells. In the abstract model the cells are randomly but densely distributed in the sheet, which means that the cells are irregularly shaped and fill the space of the sheet. The dots on the simulation pictures can be thought of as the nuclei or centers of the cells. I assume that cells can not arbitrarily overlap. In the simulator this is implemented by choosing a minimum distance between cells. The minimum distance is a fraction of the distance that would exist between cells if they were placed on a regular grid. The simulator places each cell randomly in the plane, however if it is too close to another cell the simulator attempts to place it again at most 50 times. After that it simply places it anywhere.

Initial State: The cells in the sheet start out with a few simple initial conditions, which are the same for all simulations. The sheet has an apical and basal side, which means that all cells have internal apical/basal polarities that point in the same direction. The simulation pictures are as seen from the apical side of the sheet. For most simulations the sheet is assumed to be square. Cells know if they belong to one of the edges of the sheet, and which edge they belong to. An edge of the sheet is defined as the region less than the communication radius r away from the actual sheet boundary. Note that a cell has only one bit of information per edge — there is no access to global coordinates or position within the edge.

Simulating Folding: As mentioned before, the abstract model restricts the possible operations on a sheet to straight flat folds. This restriction greatly simplifies the simulation of the cell sheet. An individual cell calls `local-fold` within its program with two arguments: a local orientation (unit vector) and a surface (apical or basal). These arguments loosely represent the fibers the cell would have contracted in a dynamic model. Given a set of such cells, the simulator determines whether or not the result can be approximated by a straight flat fold on the sheet, and if so, computes the new configuration of the sheet after the fold. The criteria for determining whether or not to fold the sheet are adapted from the dynamic model. There must be a sufficient number of cells along the width of the crease and the width of the crease must be reasonably uniform. The crease can not be crooked or discontinuous because that would produce wrinkles or complex deformations that can not be modeled. The simulator must also confirm that the cells on average can correctly predict the fold orientation and surface. After checking these criteria, the simulator performs the fold along the best-fit-line that minimizes perpendicular deviations from all the cells in the crease. In chapters 4 and 6 the criteria are discussed in more detail. The new configuration of the sheet is easily computed by reflecting all the cells on one side of the fold. The simulator keeps track of the layers of cells.

Sensing Contact: The simulator also implements the sensing capability of the cells. Cells in direct contact with each others apical or basal surface are considered part of each others local communication neighborhood. The simulator computes which cells are in contact with each other each time the configuration of the sheet is changed. For determining contact the cell has a size proportional to the radius it would have if all the cells were disks packed in a grid. If this were implemented exactly then each cell would essentially grow out from its nucleus and push against other cells, resulting in a irregular shape for each cell. Instead we assume this simpler circular shape and allow for some overlap in sensing as well as the possibility of holes in sensing.

The next chapter discusses a global paradigm for creating shapes from a sheet by folding.

Chapter 3

The Origami Shape Language (OSL)

In the previous chapter I introduced my model for a programmable material — a sheet of cells that can fold. Here I present a way of thinking about creating shapes by folding. Origami can be considered as a language for constructing global shape from a continuous sheet. The construction tools are a set of folding techniques. A shape is described as a sequence of straight folds performed on a sheet of paper, with no cuts or glue. The Origami Shape Language is based on this notion of describing shape as a folding construction. This chapter presents the Origami Shape Language.

3.1 Introduction to Origami Mathematics

Origami is an ancient art form, probably as old as paper itself. With origami one can produce models of significant (and often unbelievable) complexity — from the traditional crane, to many-appendaged realistically-proportioned insects and three dimensional polyhedra [61, 37, 52]. What most people are not aware of is the deep relationship between origami and traditional geometry.

In the past two decades there has been a renaissance in the mathematics of origami. A seminal paper by Humiaki Huzita in 1989 first related origami to plane Euclidean constructions, also known as straight-edge and compass constructions [31]. Huzita provided a set of 6 axioms that describe the construction of most origami folds. If one thinks of folding as a tool for creating lines on a sheet of paper, the "crease-pattern" can be compared to other traditional 2D constructions. He proved that the first four axioms are equivalent to plane Euclidean constructions and that the sixth axiom is more powerful and can solve polynomials of degree three (e.g, angle trisection, cube doubling) [30, 20]. Since then there has been an explosion in the understanding of the mathematics behind origami constructions.[1]

Several results show what types of shapes can be constructed from a sheet using

[1] A great reference for origami mathematics is a website by Professor T. Hull of Merrimack College who also teaches a course on computational geometry using origami [29].

origami and provide automatic techniques for deriving the folding sequence and crease patterns. Robert Lang, an accomplished origamist and laser physicist, proved that the construction of all tree-based origami (e.g. the basic floor plan for any animal-like shape) could be automatically derived by a computer. This work revealed many interesting underlying relationships between tree shapes and disk packing in a plane [41, 45]. Demaine *et al.* proved that all 2D polygonal regions, including ones with holes, could be constructed using a small set of origami operations; they are currently exploring relationships between planar surfaces and convex and concave polyhedra. There are several important theorems that provide sufficiency conditions for determining whether creases that meet at a point can be folded flat; however, determining whether an arbitrary crease pattern folds flat is NP-hard [29, 38, 9]. The space of 3D structures is only beginning to be studied, but expert origamists recognize few if any limitations on what can be created from a single piece of paper [37].

Concepts from origami have also been applied to many practical applications. A famous example is the *miura-ori* fold by Koryo Miura, designed for deploying large solar panels [50]. The design, based on origami folding, consists of a paneled surface that can compactly fold into a small space but then easily unfold without jamming and without deforming any of the panels. More recently, a large space telescope (25-100m) based on Fresnel lenses is being designed at Lawrence Livermore Labs, using a different origami fold design by Robert Lang, where again the issue is compact storage and easy non-deforming unfolding [32, 33]. There are many other applications of origami to both science and manufacturing: relating stiffness properties created by origami patterns to buckling tubes, studying folding structures in nature (folding and unfolding wings, unfolding leaf buds), manufacturing single sheet cups and boxes and folding maps [40, 22, 51, 26].

This work presents a very different application of origami, but it benefits directly from the large body of work on origami math and origami design. The Origami Shape Language is based on Huzita's axioms.

3.1.1 Huzita's Axioms of Origami

The Italian-Japanese mathematician, Humiaki Huzita, has described a set of 6 axioms for constructing a new crease (or line) from a set of points and line on a sheet (figure 3-1).

A1. Given two points `p1` and `p2`, fold a line through them.

A2. Given two points `p1` and `p2`, fold `p1` onto `p2` (constructs the crease that bisects the line `p1p2` at right angles).

A3. Given two lines `L1` and `L2`, fold `L1` onto `L2` (constructs the crease that bisects the angle between `L1` and `L2`).

A4. Given `p1` and `L1`, fold `L1` onto itself through `p1` (constructs a crease through `p1` perpendicular to `L1`).

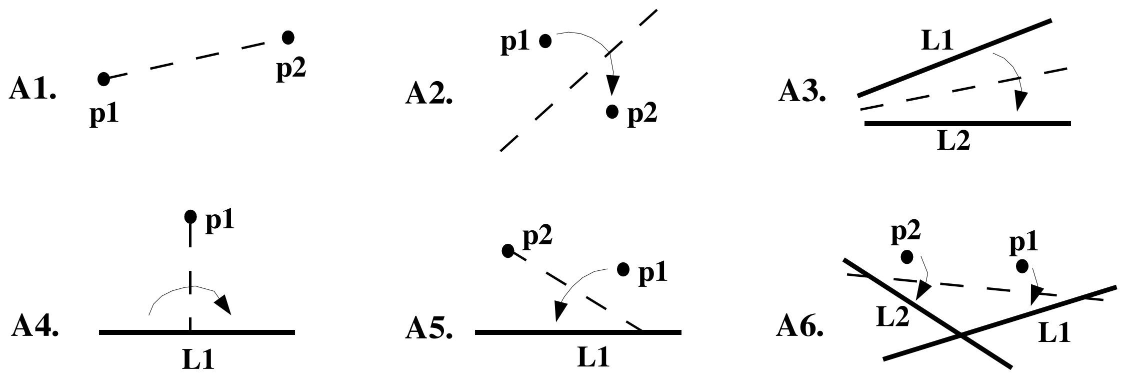


Figure 3-1: Huzita's axioms of origami

A5. Given `p1` and `p2` and line `L1`, make a fold that places `p1` on `L1` and passes through `p2` (constructs the tangent to the parabola (p1 L1) through p2)

A6. Given `p1` and `p2` and lines `L1` and `L2`, make a fold that places `p1` on `L1` and `p2` on `L2` (constructs the tangent to two parabolas).

Most complex origami folds can be reduced to a sequence of these axioms. Axioms 1-3 are the most commonly used and in this thesis I only use the first four axioms. Each axiom can be thought of as creating a line on a sheet, by first folding and then immediately unfolding. New points are created by the intersection of lines. The axioms only describe how to find the location of a new crease but not how to fold it. They do not capture several ideas that are implicit in origami diagrams such as the type of fold or the number of layers of paper to fold through.

3.2 Origami Shape Language Specifications

In the Origami Shape Language (OSL) the shape is described as a sequence of operations on a sheet. The syntax of the language is based on Scheme [2, 24]. This section first describes the OSL language: the primitives, the means of combination and the means of abstraction. Then the language is illustrated through several examples.

Basic Elements: Points, lines and regions.

Initial Conditions: A sheet with a set of defined boundary lines and points (edges and corners). The sheet does not have to be square though for most of the examples it is assumed to be. The sheet also has an apical (top) and basal (bottom) surface.

Primitive operations:

```
(crease-lbp p1 p2 c)            ; line between points
(crease-p2p p1 p2 c)            ; point to point
(crease-l2l l1 l2 c)            ; line to line
(crease-l2s l1 p1 c)            ; line to itself
```

These are invocations of Huzita's axioms 1 through 4. The language only includes the first four axioms. The return value for each of these operations is a new line. `c` is optional and specifies a color for the crease line to be displayed by the simulator.

```
(intersect l1 l2)
```

Returns a new point that is the intersection point of the two lines.

```
(execute-fold l1 type landmark=p1)
```

Execute-fold folds the sheet along line `l1`. The `type` of the fold is either apical or basal. In origami there are two types of folds, mountain and valley, that make use of an implicit top surface (figure 3-2). Diagrams always draw the top surface facing the viewer. The type of fold is relative to this top surface — a valley fold cause the top surface to touch itself while a mountain fold causes the bottom surface to touch itself. After a fold, the new top surface is ambiguous. In origami diagrams the new top surface is chosen by using an arrow to show which side "moves" to the other when a fold is executed. The top surface is made explicit in OSL by having the sheet maintain an apical and basal surface. There are two types of folds, apical and basal, that correspond to the valley and mountain folds. For a given crease, a fold that puts the apical surface on the inside is an apical fold and vice versa. After the fold is executed, the apical surface must be re-determined. One side of the fold must reverse its apical/basal polarity for the new sheet to have an apical and basal surface as before. This side is chosen using a `landmark`, which is a point or line on the side of the fold that will reverse its polarity. The fold is always a flat fold, and hence the structure created is a flat but layered structure, also called *flat origami*. A very large segment of origami falls in this category.

```
(create-region p1 l1)
(within-region r1 op1 ...)
```

`create-region` returns a region. A region is defined by a crease line `l1` that divides the sheet into two disjoint parts and a point `p1` which indicates which part is the region `r1`. `within-region` restricts any of the above operations to occur only within the defined region `r1`. Regions allow us to capture several important ideas. Most important is the concept of partial layer folds. Folds are assumed to go through all the layers of the sheet. However in origami it is not uncommon, and is very important, to make folds through only some layers. By defining a region and later on executing commands within that region (including folds) one can execute folds through partial layers. In origami diagrams the extent of a fold can be very difficult to decipher but the regions make this unambiguous. Regions also provide the ability to refer to line segments and easily program substructures.

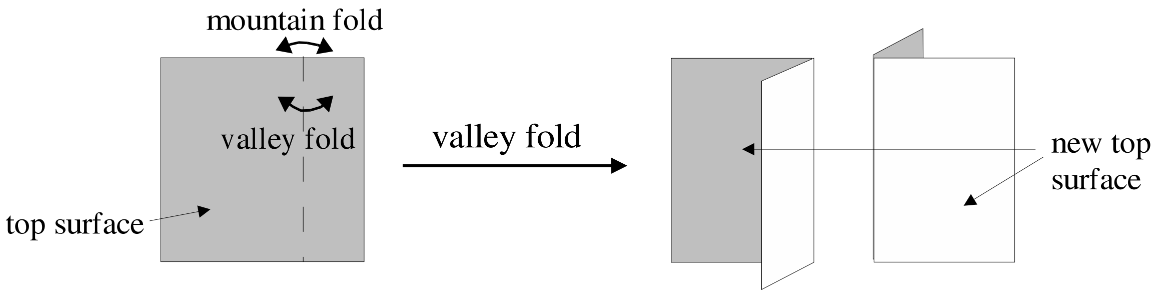


Figure 3-2: An origami fold can be either a valley fold or mountain fold relative to the current top surface. After the fold, a new top surface must be chosen and there are two choices.

```
(or a ....)
(not a)
```

Ability to define complex collections of points, lines and regions.

```
(define name op1)
```

Ability to give names to the results of operations.

Combination: The program is written as a sequence of operations on the sheet. The sheet starts with the initial lines and points (edges and corners). Each operation creates new lines and points that can be named using define and then used in subsequent operations. For example

```
(define d1 (crease-p2p c3 c1))
(define d2 (crease-l2l e23 d1))
(define p1 (intersect d2 e34))
(execute-fold d1 apical landmark=c3)
.....
```

Expressions can be nested, however the order in which the arguments are evaluated is not specified, therefore the end result must not depend on the order. In most origami constructions the nesting is limited by the dependencies between folds.

Abstraction:

```
(defun (name arg1...) op1 ...)
```

The main form of abstraction in the language is through procedures. Procedures are mainly used to capture the construction of repeated structures in an origami model, for example the legs of a insect, or common bases (common starting fold sequences). Any names defined within a procedure are local to the procedure and can not be seen outside. A procedures can return a value. Currently the compiler implements only simple procedures, but one could incorporate more general aspects of a high-level language, like branching statements and/or recursion. Another source of abstraction and modularity comes from origami itself. Many different structures are constructed from common bases and common folding sequences. Maekawa has described a set of small "molecule" crease patterns that create particular substructures and he designs new structures by using tessellations of these molecules [45].

Using this language one can translate an origami diagram into a program. The following example, a folded cup, illustrates how this is done.

3.2.1 Example: A Cup

The following cup is an interesting example of a functional flat layered shape. It has a very simple folding sequence, but at the same time illustrates many of the subtle points of origami. Here I show how a cup is constructed from a square sheet and how this is expressed using OSL. In the next chapter we revisit this example to see how the cells in a sheet coordinate to assemble a cup.

The folding diagram for the cup is shown in figure 3-3 and the corresponding OSL program is shown in figure 3-4. The square sheet starts out with four corner points (`c1, c2, c3, c4`) and four edges (`e12, e23, e34, e45`) that define the boundary. First we construct the diagonal `d1` from the points `c1` and `c2` using axiom 2 (`crease-p2p`). The diagonal divides the sheet into two regions and we name these regions `front` and `back`. We will use these regions later. An apical fold is executed along the `d1` line. Next we create the line `d2`. Not all lines are folded, for example the line `d2` is an intermediate step, used only to find the location of point `p1`. The crease `d3` is created by folding the corner `c2` to `p1`. For the `execute-fold` on `d3`, the landmark chosen is `c2` which ensures that after the fold, the apical surface is the side with the flap on top. The choice of landmarks is important and ensures that both flaps of the cup end up on the same side. All the folds go through every layer of the sheet, except for the last two. The line `l1`, created using axiom 1, goes through both layers of the sheet, however we want to fold the front layer towards us and back layer away from us. Here is where we use the regions; an apical fold is executed in the `front` region and a basal fold in the `back` region. In this case single layers were folded but the same idea could be used for multiple layers.

In the end if we open up the sheet we get a crease pattern that tells us the location of all the creases. An interesting fact is that the crease pattern does not encode sufficient information to reconstruct the shape, because it does not encode the order of folds. For example if `l1` is folded before `d1` or `d3`, the cup will fail.

The language does not prevent one from writing nonsense, i.e. trying to fold something that can not be done. It is up to the simulator to implement controls to try to prevent the physically impossible. The next section presents several other examples of OSL code.

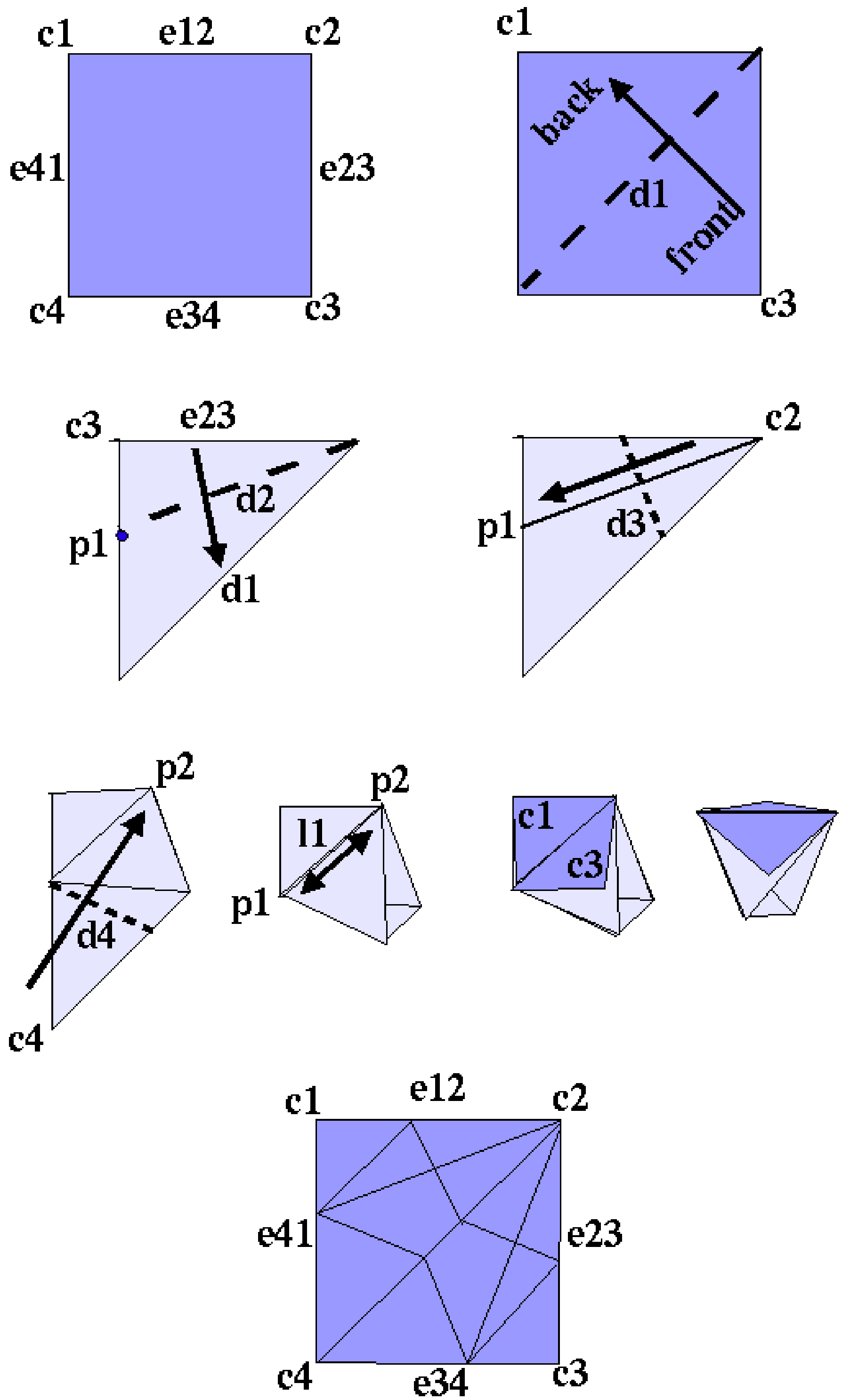


Figure 3-3: Folding diagram for a cup

```
;; OSL Cup program
;;---------------------

(define d1 (crease-p2p c3 c1))
(define front (create-region c3 d1))
(define back  (create-region c1 d1))
(execute-fold d1 apical landmark=c3)

(define d2 (crease-l2l e23 d1))
(define p1 (intersect d2 e34))
(define d3 (crease-p2p c2 p1))
(execute-fold d3 apical landmark=c2)

(define p2 (intersect d3 e23))
(define d4 (crease-p2p c4 p2))
(execute-fold d4 apical landmark=c4)

(define l1 (crease-lbp p1 p2))
(within-region front
  (execute-fold l1 apical landmark=c3))
(within-region back
  (execute-fold l1 basal landmark=c1))
```

Figure 3-4: OSL program for a cup

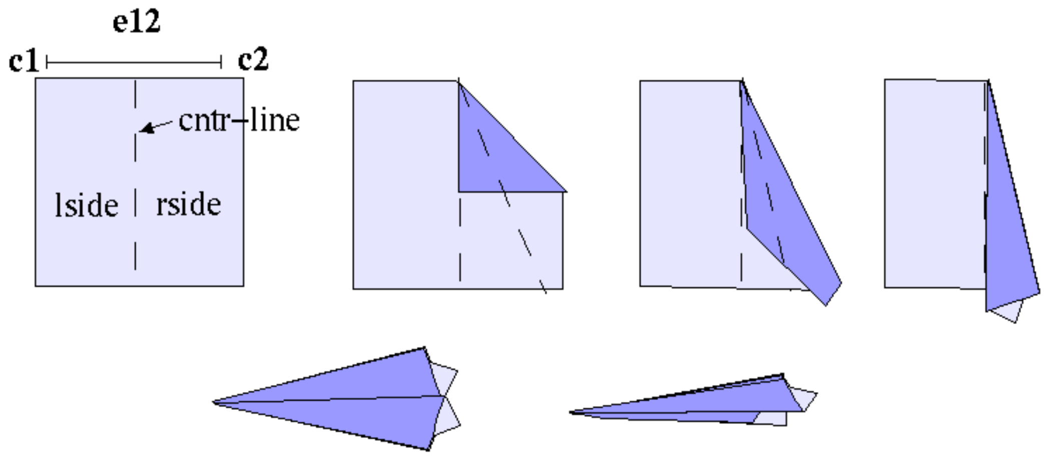


Figure 3-5: Folding an airplane

3.2.2 Other Examples

Airplane

This is the OSL program for folding a paper airplane. In this example both wings of the airplane are symmetric and can be captured by a single procedure `fold-wing`. We use regions to divide the line `e12` into two segments. In the first operation of the procedure, `e12` is folded onto `cntr-line`. When fold-wing is called within the region `rside` only the right half of `e12` is used.

```
(define cntr-line (crease-l2l e14 e23))
(define rside (create-region c2 cntr-line))
(define lside (create-region c1 cntr-line))

(defun (fold-wing cntrline topline ucorner side dcorner)
    (define t2 (crease-l2l topline cntrline))
    (execute-fold t2 apical landmark=ucorner)
    (define t3 (crease-l2l t2 cntrline))
    (define p3 (intersect t2 side))
    (execute-fold t3 apical landmark=p3)
    (define t4 (crease-l2l t3 cntrline))
    (execute-fold t4 apical landmark=dcorner)
    )

(within-region rside (fold-wing cntr-line e12 c2 e23 c3))
(within-region lside (fold-wing cntr-line e12 c1 e14 c4))
(execute-fold cntr-line basal landmark=c4)
```

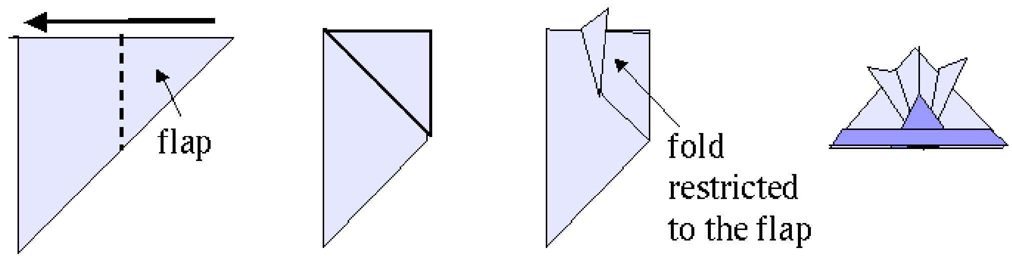


Figure 3-6: Folding a samurai hat

Samurai Hat

The samurai hat uses folds that go though partial layers of the sheet. In origami there is an informal notion of a flap — a region of the sheet joined to the main body by a single crease. This is used to describe partial layer folds. The code below shows part of the samurai hat code. First the sheet is folded into a right angle triangle. Then one of the corners on the hypotenuse (`corner`) is folded onto the right-angle vertex `c1`, creating the line `d2`. If we think of the sheet between the line `d2` and the corner as a flap, the next fold is executed only on this flap. In the code we use regions to express this idea.

```
(define d1 (crease-p2p c1 c3))
(execute-fold d1 apical landmark=c1)

(defun (fold-hat-side corner edge)
    (define d2 (crease-p2p corner c1))
    (define flap1 (create-region corner d2))
    (execute-fold d2 apical landmark=corner)

    (within-region flap1
        (define d3 (crease-l2l edge d2))
        (define d4 (crease-l2l d1 d3))
        (execute-fold d4 apical landmark=corner)
        ))

; fold right and left sides of the hat
(fold-hat-side c2 e12)
(fold-hat-side c4 e14)

; fold front and back bottom of the hat
..........
```

Grid

The Origami Shape Language can also be used to create patterns of lines on the sheet surface. For example, the following code produces a 4×4 grid on the sheet.

```
(define h1 (crease-l2l e34 e12))
(define h2 (crease-l2l h1 e12))
(define h3 (crease-l2l h1 e34))
(define v1 (crease-l2l e23 e14))
(define v2 (crease-l2l v1 e23))
(define v3 (crease-l2l v1 e14))
```

In chapter 5 we explore in depth what kinds of shapes and patterns can be expressed using OSL and self-assembled by the cell sheet.

3.3 Why Origami?

There are many other choices for describing a shape other than origami, such as a set of polygon layers. The most attractive feature of origami is that one can construct a wide variety of complex shapes using a few axioms, simple fixed initial conditions and one mechanical operation (a fold). Origami has considerable descriptive power. Huzita has proven that the axioms can construct all plane Euclidean constructions. Lang has shown that all tree-based origami shapes can be automatically generated by computer and Demaine *et al.* provide a method for constructing scaled polygonal shapes [41, 15]. Not all of these can be expressed using Huzita's axioms, however there is a large literature of shapes that can [37]. As the relationship between origami constructions and geometry is explored further, the results will directly impact this work.

This suggests an advantage to using a *constructive* global description. Rather than trying to map the desired end goal directly to the behavior of individual elements, the problem is broken up into two pieces: a) how to construct the goal globally and b) how to map the construction steps to local rules. By doing so, we can take advantage of knowledge from other disciplines of how to decompose a problem. This highlights one of the hard problems in thinking about self-organizing systems — finding the right global description.

Chapter 4

The Biologically-inspired Cell Program

In this chapter I describe how the cell program is automatically derived from the global shape description in OSL. All cells have the same program and differ only in a small amount of dynamic local state. A cell executes the program autonomously based on local communication with its neighbors. The paradigm at the cell level is inspired by developmental biology.

The compilation process is described in three steps: First, I present a small set of biologically-inspired primitives that form the basis of how the cells organize and self-assemble. Then I show how each of the operations in OSL, such as the axioms, can be implemented by simple cell programs. Finally I show how a shape description is compiled into a cell program. The final cell program uses only a small amount of local state and the majority of the program is conserved across all shapes.

4.1 Biologically-inspired Local Primitives

In this section, I present a small set of biologically-inspired primitives. The primitives are simple ways in which a cell interacts with its local neighborhood and uses its sensing/actuation capabilities. The primitives are inspired by studies of embryogenesis in simple multicellular organisms such as the *Drosophila* and sea urchin [42, 63, 72]. There are five primitives: gradients, neighborhood query, polarity inversion, cell-to-cell contact, and flexible folding.

4.1.1 Gradients

Gradients are a powerful and well known idea in developmental biology [71]. Gradients usually refer to a gradient of some chemical concentration. Although gradients had been hypothesized to be instrumental in pattern formation and regulation for a long time, a famous breakthrough came when Nusslein-Volhard and Wieschaus discovered gradients of proteins in the *Drosophila* embryo. These gradients determine the initial segmentation into head, thorax and abdominal regions, as well as the

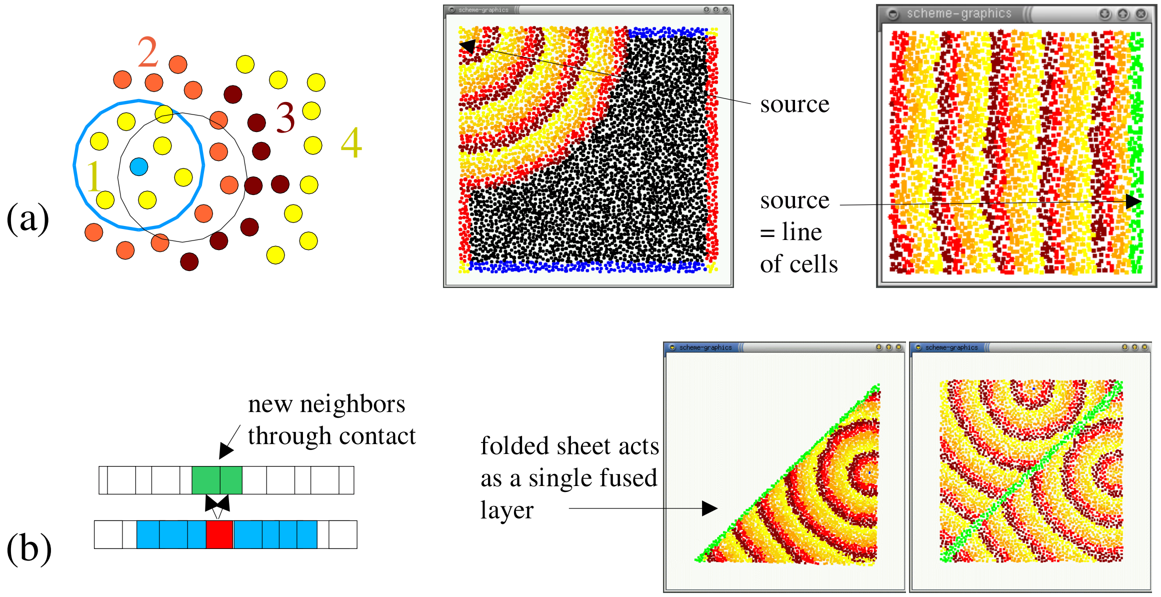


Figure 4-1: (a) Gradients from a single cell and a line of cells (b) Cell-to-cell contact changes the local neighborhood of a cell. This affects the way gradients propagate.

dorsal-ventral axis [55, 42].

The gradient primitive is analogous to a chemical concentration that increases in value as one moves away from the source (figure 4-1(a)). A cell creates a gradient by sending a message to its local neighborhood with a gradient name and a value of one. The gradient is propagated through local communication. The neighboring cells forward the message to their neighbors with the same name and the value incremented by one. Their neighbors forward the message in the same way, and this continues until the gradient has propagated over the entire sheet. Each cell stores the minimum value it has heard for any particular gradient name. In a strictly topological sense this is equivalent to generating a breath-first-search tree (BFS) [44]. However because of the spatial locality of communication, the gradient also provides a rough estimate of distance and direction relative to the source.[1] Cells average the gradient value over their neighbors to improve the distance estimate.[2]

Many cells can emit a gradient with the same name, in which case the gradient value reflects the shortest distance to any of the sources. Thus, the shape and positions of the sources affects the spatial pattern of gradient values. For example if a single cell emits a gradient then the value increases as one moves radially away from the cell (figure 4-1(a)). If a line of cells emits a gradient then the gradient value increases as one moves perpendicularly away from the line. A cell can also create a *bounded gradient* which implies that certain types of cells will not forward the gradient message;

[1]The gradient primitive is different from reaction-diffusion and ant pheromone-like primitives that depend on more strictly modeling chemical diffusion, interactions and evaporation.

[2]Chapter 6 discusses the accuracy of gradients in detail.

the intuition being that certain cells can act as barriers to particular gradients.

In the cell program, a cell can create a gradient by calling `(create-gradient <name>)` and check it if has received a gradient value by calling `(recvd? <name>)`. The time at which a cell receives a gradient value is proportional to the distance from the source. Each cell propagates gradients in the background. A cell can use a gradient value it has received in many ways: it can compare the value to an absolute threshold, it can compare the values of different gradients, or it can collect gradient values from its neighbors and determine which neighbors are closer to the source (i.e. have smaller gradient values). Different variations of the gradient primitive have been explored within amorphous computing and elsewhere [53, 13, 27, 47].

4.1.2 Neighborhood Query

This primitive allows a cell to send a query to its local neighborhood and collect information about their state. For example a cell may collect neighboring values of a gradient. This primitive is similar to that used in cellular automata [46]. A cell can also broadcast a message to all the cells in its local neighborhood.

4.1.3 Polarity Inversion

As part of the initial conditions of the sheet, each cell has an internal apical-basal polarity. Originally all cells have the same polarity, but a cell can choose to invert its internal apical-basal polarity.

4.1.4 Cell-to-cell Contact

This primitive connects sensing with communication. When cells come into direct physical contact with each others apical or basal surface, as a result of changes in the shape of the sheet, they become part of each others local communication neighborhood. If cells cease to be in contact, then that communication bond is broken. For the most part a cell does not distinguish between the original neighborhood and the contact neighborhood. This primitive is implemented within the simulator and the contact neighbors are recalculated every time the sheet is folded.

Cell-to-cell contact is the main way in which changes to the environment (sheet) affect the behavior of the cells. It allows multiple layers of the sheet to act as a single fused layer. For example, since cell-to-cell contact affects the local neighborhoods, it indirectly affects the way gradients propagate. Gradients seep through regions of the sheet that are in physical contact with each other, as shown in figure 4-1(b).

4.1.5 Flexible Folding

This primitive comes from the actuation model of the cell described in chapter 2. A cell can fold itself along a particular orientation by calling `local-fold` within its program with two arguments: a pair of neighbors and a surface (apical or basal). The pair of neighbors represents the orientation of local folding.

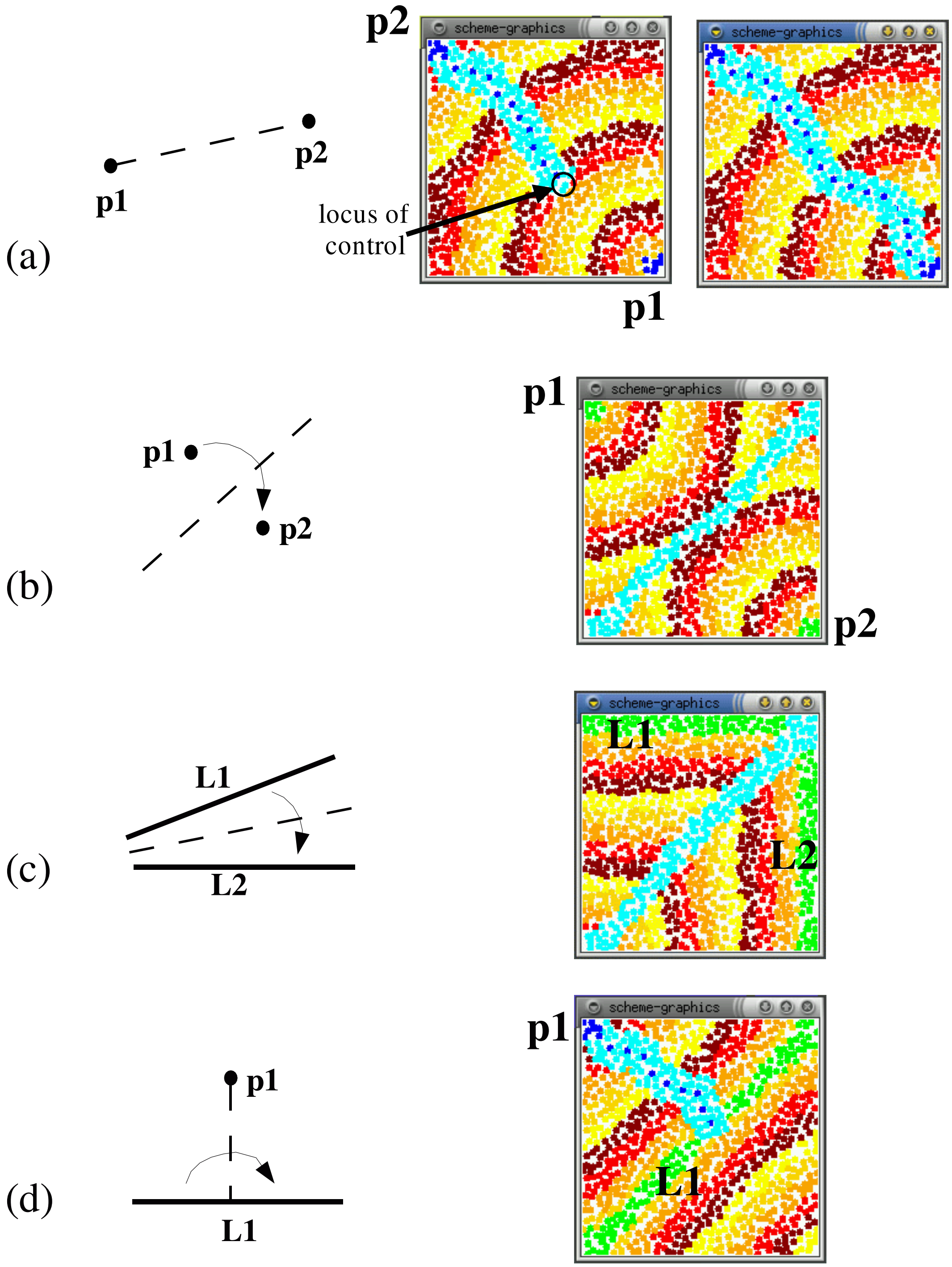


Figure 4-2: Huzita's axioms implemented by the cells: (a) crease-lbp (b) crease-p2p (c) crease-l2l (d) crease-l2self. The cells that are part of the input are shown in green and the cells in the output line are shown in cyan.

4.2 Implementing OSL Operations

This section describes how each operation in the Origami Shape Language (such as an axiom or create-region) can be implemented using the set of primitives just described. The OSL operation is achieved by all cells executing the same cell program based on their internal state. Here the programs are expressed in Scheme but they could also be expressed as finite state machines, I/O automata, Microbial Colony rules or StarLogo programs [2, 13, 67, 62].

4.2.1 Initial Sheet

Initially the sheet starts out with four distinct lines and points (edges and corners). A point is represented by a group of cells approximately the size of a local neighborhood. A line is represented by a line of cells of width approximately the diameter of the local neighborhood. All cells in a line or point group are equal i.e. no one cell is in charge of the group. Each cell has a boolean variable in its local state corresponding to each distinct point/line. The variable is set to true if the cell is part of the point/line and false if it is not. Initially all cells have boolean state variables for `e12, e23, e34, e41` and `c1, c2, c3 c4`. If a cell is part of the edge `e12` then the corresponding state variable is true, and so on. The initial conditions are set by the simulator and automatically break the symmetry of the sheet, therefore we are not concerned about symmetry breaking. The initial conditions are very simple; cells do not know where they are within an edge and the remainder of the sheet is homogeneous, just like a blank sheet of paper.

4.2.2 Axioms

The axioms make use of the fact that gradients provide an estimate of the distance to the source, as well as reflect the shape of the source.

Axiom 2 (crease-p2p): Axiom 2 states that given two points `p1` and `p2`, we can fold `p1` onto `p2` creating a line that bisects the line `p1 p2`. What this means is that any point on the new line is equidistant from both `p1` and `p2`. If the cells in `p1` and `p2` create two different gradients, every cell can simply compare the values to determine if they lie on the new line.

Figure 4-3 shows the cell program for the axiom and figure 4-2 (b) shows a simulation of this cell program on a sheet. The process by which the crease forms is as follows: First the cells in `p1` create a gradient `g1`. When the gradient arrives at the cells in `p2`, they create a gradient `g2`. When this gradient arrives back at the cells in `p1`, then these cells first wait until they are reasonably sure that the gradient has traveled across the sheet. This delay value is calibrated in the beginning of every shape formation and is discussed later. The `p1` cells then send a gradient `gend` that marks the completion of the axiom and cells can compare the values they received for `g1` and `g2`. The ending gradient plays an important role in being able to sequence operations. The cells compare whether the absolute difference between the gradient

```
(define (axiom2-rule p1 p2 g1 g2 gend)
  ; arguments are boolean state p1 p2 and gradient names
  (if p1 (create-gradient g1))
  (if p2 (begin (wait-for-gradient g1)
                (create-gradient g2)))
  (if p1 (begin (wait-for-gradient g2)
                (wait local-delay) ; calibrated local delay
                (create-gradient gend)))

  (wait-for-gradient gend)
  (if (< (abs (- g1 g2)) 2)      ; width of the crease is apprx 2r
      #t
      #f))

; To execute the program, call the procedure
(define d1 (axiom2-rule c1 c3 ga gb gc))
```

Figure 4-3: Cell program for axiom 2 (crease-p2p). The same cell program also works for axiom 3.

values is less than some threshold; the threshold determines the width of the crease formed. In this case the axiom produces a line of cells of width approximately 2 times the local communication radius r. The program is expressed as a Scheme procedure. A cell executes the program by calling the procedure with particular point and gradient names and can assign the return value to some named boolean variable.

Axiom 1 (crease-lbp): Axiom 1 states that one can create a line between two points `p1` and `p2`. In order to implement this we borrow an idea from the Growing Point Language by Coore [13] and implement it using our primitives (figure 4-2 (a)). The cells belonging to `p2` create a gradient. When the gradient arrives at `p1`, the cells in `p1` each do a neighborhood query to determine if they have the lowest gradient value. The `p1` cell with the lowest value becomes the growing point. A growing point is a locus of control that is passed from cell to cell along the direction of the gradient. At each step the current growing point cell uses neighborhood query to find the neighbor with the lowest gradient value. The cell then sends a message to that neighbor to be the next growing point. When a cell belonging to `p2` is chosen, then the line is complete. That cell sends a second gradient `gend` to mark the successful completion. Throughout the process all cells within a radius of r of the growing point become part of the crease, producing a crease of width $2r$. The idea is inspired by chemotropism in the growth of plants and neurons.

Figure 4-4 shows the cell program for axiom 1. The program uses temporary cell identifiers for exchanging messages between neighboring cells. The identifiers are generated using a random number generator such that the probability of names being locally unique is very high. The code does not depend on globally unique identifiers. Since even local uniqueness of the identifiers is not guaranteed, it is possible to end up with more than one growing point. In the rare case that this does happen, the result is a fatter crease and the axiom does not fail.

Axiom 3 (crease-l2l): Axiom 3 is implemented using the same cell program as axiom 2, by taking advantage of the fact that gradients produced by a line of cells reflect the distance from the line. Axiom 3 states that given two lines `l1` and `l2`, one can create a line by folding `l1` onto `l2`. The resulting line either bisects the angle between `l1` and `l2`, or if `l1` and `l2` are parallel then it is a parallel line midway between the two lines. What this means is that any point on the crease line is equidistant from both inputs. This is the same condition as axiom 2 and hence we can implement axiom 3 using the cell program in figure 4-3. The gradients are generated by lines of cells and hence the gradient values increase as one moves perpendicularly away from the line of cells (figure 4-2 (c)). The cells with equal gradient values lie along the angle bisector.[3]

Axiom 4 (crease-l2self): Axiom 4 creates a crease line by folding `l1` onto itself, such that the crease line goes through the point `p1`. The resulting line is a line from `p1` to `l1` that is perpendicular to `l1`. We can use the same cell program as axiom 1 to create this line. As in axiom 3, we are using the fact that a gradient from a line of cells produces values parallel to line. The cells in `p1` grow a line towards `l1` (figure 4-2(a)).

Each axiom attempts to produce a crease that is approximately twice the width of the local communication radius r. Each axiom ends with a gradient, and cells use this final gradient as a signal that the axiom has completed.

Intersect: Intersection of lines is simply an AND operation on the boolean state variables corresponding to those lines.

```
(define (intersect-rule l1 l2)
   (and l1 l2))
```

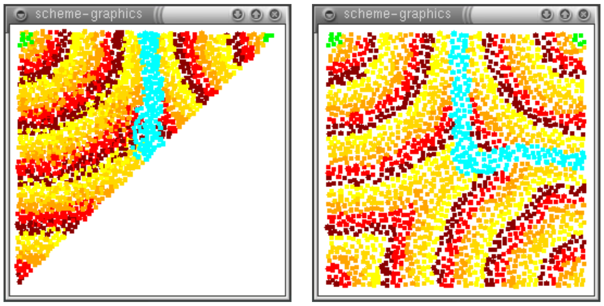
4.2.3 Seepthru and Regions

A key piece to making the axioms work is the concept of *seepthru*. Seepthru does not correspond to any global operation in OSL, rather it refers to the behavior of gradients on a folded sheet. When a sheet is folded, cells make new neighborhoods through contact and the gradients values propagate through the layers. As a result multiple layers of a sheet can act as a single layer and the gradient values reflect the

[3]With axiom 3, it is possible to get more than one answer. For example, if the two input lines intersect like an X, then two crease lines are produced. Regions can be used to isolate the cell program to one quadrant.

```
(define (axiom1-rule p1 p2 gstart gend)
 (define id (random million))     ; temporary cell identity
 (define gpt #f)                  ; gpt = growing point
 (define crease #f)

 ; cells in p2 create the gradient
 (if p2 (create-gradient gstart))

 ; p1 cells compete to start growing the crease
 (if p1 (begin (wait-for-gradient gstart)
              ; the p1 cell with the lowest gstart value starts
              (set! new (find-best1 ;*, include self
                           (cons (nbrhood-query (list id gstart p1))
                                 (list id gstart #t))))
              (broadcast-msg ('next-gpt new))))

 (let loop ()
   (check-for-messages-or-gradient)
   (if (recvd-msg 'next-gpt)
     (if (and (not p2) (= id msg-arg)); if chosen as next growing pt
       ; collect nbr values, and choose one with lowest gstart value
       (begin (set! new (find-best2 ;*
                          (nbrhood-query (list id gstart (not gpt)))))
              (broadcast-msg ('next-gpt new))  ; send a message
              (set! gpt #t)))

     (if (and p2 (= id msg-arg))     ; crease has reached p2
       (begin (create-gradient gend) ; send a completion gradient
              (set! gpt #t)))

     (if (not (= id msg-arg))  ; all cells within broadcast radius
       (set! crease #t)))      ; of the growing pt are in the crease

   (if (recvd? gend) ; axiom is complete
     ; cell is part of the crease if gpt or neighbor of gpt
     (or gpt crease))
 ))

;; * find-best1 and find-best2 are just simple list filters
;; find-best1 returns the id of the p1 cell with the smallest gstart
;; find-best2 returns the id of the non-gpt cell with lowest gstart
```

Figure 4-4: Cell program for axiom 1 (crease-lbp). The same cell program also works for axiom 4.

(a) 

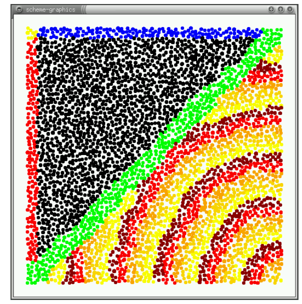
(b) 

Figure 4-5: (a) Seepthru allows a folded sheet to acts a single layer. The crease formed by axiom 2 goes through both layers. (b) A region is formed by using a bounded gradient. The green cells form an impassable boundary.

Euclidean distance from the source, rather than the distance along the plane of the sheet. This allows all of the previously defined cell programs to work as before (figure 4-5(a)). Seepthru works by changing the *context* in which the cell program applies.

Regions allow the user to *restrict the context* in which the cell program applies. A region is defined by a crease line `l1` that divides the sheet into two disjoint regions and a point `p1` on one side of the crease. The cells in `p1` create a bounded gradient that can not pass through the cells in `l1`. The cells that receive the bounded gradient become part of the region (figure 4-5(b)). The following cell program is for creating a region.

```
(define (create-region-rule p1 l1 gbounded gend)
  (if p1 (begin (create-bounded-gradient gbound l1))
                ; bounded by l1 cells
                (wait local-delay)
                (create-gradient gend))) ; unbounded

  (wait-for-gradient gend)
  (if (recvd gbounded)
      #t
      #f))
```

`Within-region` affects all of the cell programs. Each cell has a boolean state variable called `current-region` which is true or false depending on whether the cell is part of the specified region. If a cell is not part of the specified region, it does not create any gradients and automatically returns false for any cell program. It just waits for the final gradient in each operation to know when the cell program has ended. All cells still propagate and receive gradients normally.

4.2.4 Execute-fold

The last important global operation is `execute-fold`. The cells that are part of the line being folded all call `local-fold` and the simulator implements the result of the local actuation. Here I describe how `execute-fold` works; the cell program is presented in figure 4-7.

The first part of `execute-fold` has to do with maintaining an apical and basal surface of the sheet. As mentioned before, the sheet starts out with an apical and basal side (i.e. all cells have polarity pointing same way). After a fold, the apical-basal surface must be re-determined; the cells on one side of the sheet must invert their polarity for the sheet to have a consistent apical and basal surface after the fold. The landmark specifies which side of the sheet will invert polarity. Before the crease cells start contracting, the landmark cells use the same idea as create-region to mark the cells on its side of the line. Once that is complete, the cells in the crease start folding. When the fold is complete then the marked cells invert their polarity.

In order for the crease cells to fold, they must first locally determine which orientation to fold. This implies that the cells must locally try to determine the direction of the crease. This is done in two steps. First each cell determines the fraction of its neighbors that are part of the crease, which I call the strength of the crease. The strength is highest along the center of the crease and less towards the edges. Each cell then collects the strengths of its neighbors and chooses the two neighbors with the highest and lowest values to represent the orientation of the fibers that must be contracted. This orientation is perpendicular to the crease. Figure 4-6 shows what the locally determined orientations look like. Chapter 6 investigates how well the cells are able to predict the orientation of the fold.

The actual fold is implemented in the simulator, as described in section 2.4. Each cell requests the simulator to perform an apical or basal fold (relative to the cell's internal polarity) and gives the simulator its locally estimated orientation. The simulator computes the best-fit-line and determines that whether the width of the crease is relatively uniform about the best-fit-line and that there are no large kinks or discontinuous regions in the crease. The simulator also performs several other checks: it checks if the majority of cells agree which surface to contract and it compares the average of the local orientations to the best-fit-line.[4] Chapter 6 provides a detailed analysis of how well the axioms are able to find the expected line. When the folded configuration has been computed, the simulator sends a completion event to the cells. In a dynamic sheet the crease cells can sense when their shape has stabilized and can use a gradient as well as calibrated estimate of folding time to signal completion.

[4]The best-fit-line represents the ability of the axioms to correctly predict the position of the fold. The locally determined crease orientations represent the ability of the cells to correctly determine the direction of the crease. These two are kept separate in order to allow separate evaluation.

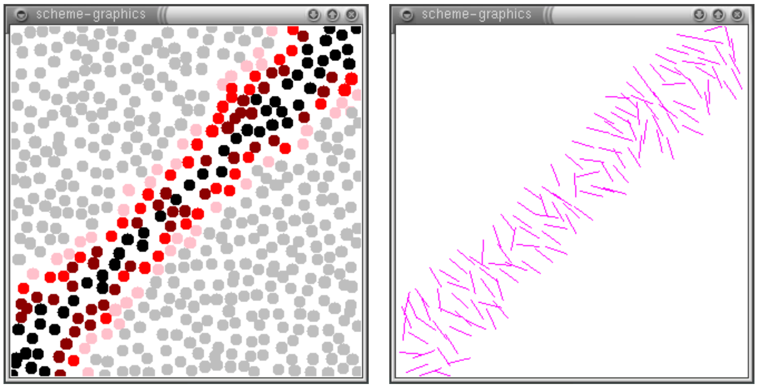


Figure 4-6: Locally determining crease direction in `execute-fold`

```
(define (execute-fold-rule l1 type p1 gbounded gend)
  (define strength 0)                ; temporary variables
  (define nbr1 0) (define nbr2 0)
  (define tmplist '())

  ; the landmark creates a region bounded by l1
  (if p1 (begin (create-bounded-gradient gbound l1)
                (wait local-delay)
                (create-gradient gend)))

  (wait-for-gradient gend)
  (if l1
      (set! tmplist (nbrhood-query (if l1 0 1)))
      (set! strength (/ (reduce + 0 tmplist) ; count nbrs in l1
                        (length tmplist)))   ; divide by total nbrs
      (set! tmplist (nbrhood-query strength))
      (set! nbr1 (max tmplist))
      (set! nbr2 (min tmplist))
      (local-fold type nbr1 nbr2) ; actuation of fibers
      )

  (wait-for-fold-end)       ; wait for event marking end of the fold

  (if (recvd? gbound)        ; if in landmark region
      (invert-polarity))    ; invert apical basal polarity
  )
```

Figure 4-7: Cell program for `execute-fold`

4.3 Compilation from Global to Local

Now that each OSL operation can be achieved using cell programs, a OSL description of a shape can be compiled into a cell program. Compiling an OSL program simply involves creating local state variables for each distinct point and line and then translating each OSL operation into a call to the corresponding cell procedure with the appropriate arguments. The compiler assigns different gradient names for each call. Gradient names can be reused by flushing old values. For procedure calls and nesting, the local state has some temporary boolean variables that can be reused when the call is complete. Figure 4-8 shows the cell program produced by compiling the OSL cup program from figure 3-4. The cell program mirrors the original OSL cup program. However note that at the OSL level there is notion of gradients, or even cells.

4.3.1 Asynchronous Operation

There is no global clock and the cells do not operate in lockstep. In an OSL program there are interdependencies between the formation of creases — the output of one axiom can be the input to the next. It is important that a cell start the next operation only when it is reasonably sure that the previous crease is complete. This is a form of barrier synchronization [44]. Since the gradients already travel over the whole sheet, they form a natural choice for a barrier. As mentioned already, in each of the cell programs there is a final gradient that marks completion. In axioms 1 and 4, the completion of the crease is very clear; `p2` sends a completion gradient when the growing crease reaches it. In axioms 2 and 3, `p1` is in charge of sending the completion gradient, however the crease formation is distributed. Therefore the `p1` cells use a locally calibrated estimate of the time taken for a gradient to travel across the sheet to assume that crease is complete. The calibration is always performed in the beginning of the cell program. A gradient is sent from `c1` to `c3` and back, while each cell uses its local clock to estimate the round-trip time taken by the gradient, as shown in figure 4-9.

Gradients take time to propagate, therefore there is always a time lag across the sheet in some direction. However because each operation ends with a gradient traveling across the sheet, this limits the maximum time difference across the sheet. Gradients themselves propagate asynchronously. A cell receives multiple messages for a gradient and finalizes the value when no more messages have been heard for some time. Because nearby cells have similar gradient values (i.e. similar shortest path lengths) they tend to remain close in time. Therefore although there is a lag across the sheet, cells tend to be in sync locally.

We assume that the time taken for the sheet to fold is significantly higher than the time taken for information to travel across the sheet. The simulator allows a fixed amount of time for cells to indicate that they want to fold before computing the folded sheet. This time is again bounded by the expected time taken by a gradient to travel across the sheet. The simulator sends all cells an event marking the completion of the fold, but in a dynamic model the landmark could be chosen to send the final gradient to mark the end of the fold.

```
;; CUP CELL PROGRAM

;; local state
;; initial state e12-e41, c1-c4, current-region=#t
(define d1 #f) (define d2 #f)
(define d3 #f) (define d4 #f)
(define p1 #f) (define p2 #f)
(define l1 #f)
(define front #f)
(define back #f)

(set! local-delay (calibrate))   ; calibration phase

(set! d1    (axiom2-rule c3 c1 g1 g2 g3))
(set! front (create-region c3 d1 g4 g5))
(set! back  (create-region c1 d1 g6 g7))
(execute-fold-rule d1 apical c3 g8 g9)

(set! d2 (axiom2-rule e23 d1 g11 g12 g13))
(set! p1 (intersect-rule d2 e34))
(set! d3 (axiom2-rule c2 p1 g14 g15 g16))
(execute-fold-rule d3 apical c2 g17 g18)

(flush g1-g15)             ; reuse gradient names, flush old values

(set! p2 (intersect-rule d3 e23))
(set! d4 (axiom2-rule c4 p2 g1 g2 g3))
(execute-fold-rule d4 apical c4 g4 g5 g6)

(set! l1 (axiom1-rule p1 p2 g7 g8))
(set! current-region front)              ; (within-region front ...)
(execute-fold-rule l1 apical c3 g9 g10)
(set! current-region #t)
(set! current-region back)               ; (within-region back ...)
(execute-fold-rule l1 basal c1 g12 g13)
(set! current-region #t)
```

Figure 4-8: The cell program compiled from the OSL cup program in figure 3-4.

```
(define (calibrate)
   (if c1 (create-gradient gc1))         ; send gradient from c1 to c3
   (if c3 (begin (wait-for-gradient gc1) ; and back
                 (create-gradient gc2)))
   (if c1 (begin (wait-for-gradient gc2)
                 (create-gradient gc3)))
   (wait-for-gradient gc1)
   (define time (clock))       ; cell's local clock
   (wait-for-gradient gc3)
   (set! time (- (clock) time)) ; measure round-trip time
   time)
```

Figure 4-9: Calibration phase. Gradients are sent from c1 to c3 and each cell locally measures the round-trip time.

4.3.2 Analysis of Resource Consumption

The resource requirements per cell are surprisingly small — a small local state and mostly fixed code space.

Resources	Per Cell
Local State	Boolean per unique point, line, region Booleans for temporary variables (procedures)
Gradients	Many used, approximately 3 per operation Short-lived, storage for no more than 6 gradients Temporary storage for gradient comparisons in axiom 1
Program Space	Mostly fixed + epsilon Fixed part corresponds to cell programs and primitives Epsilon corresponds to the shape sequence

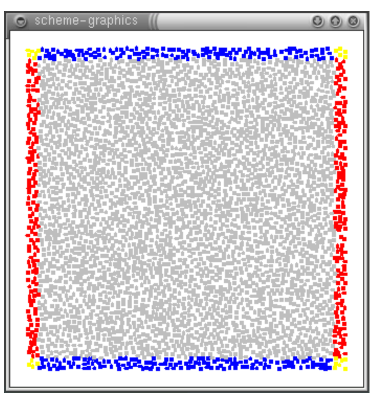
The self-assembly process is communication-intensive and relies heavily on gradients. Each axiom creates a new set of gradients. Although many gradients are created, their use is short-lived. Each operation (axiom, create-regions, etc) uses no more than 3 distinct gradients. Therefore a cell does not need to store more than 6 distinct gradients at any time; the previous set of gradients is kept around just in case neighbors are still in the previous cell program. The storage per gradient however is proportional to the diameter of the sheet because the gradients must be able to travel the entire length of the sheet.

Also, as the code in 4-8 shows, the gradient names themselves can be reused. A cell simply flushes any values it had associated with that name. This is akin to using the same chemicals or proteins over and over again, and this analogy is explored in more detail in chapter 7. Gradients are communication intensive and also affect the time taken to form the shape. Most gradients travel over the whole sheet, but as the sheet gets folded smaller the gradients have less distance to travel. Although it is possible to smart compile to reuse some gradient values across axioms, this has limited benefit because `execute-fold` changes the behavior of the gradient and therefore gradients can not be reused beyond an `execute-fold`. A more effective way of reducing the communication-intensiveness is through regions. Although not currently implemented, it is possible to bound all gradients when `within-region` is in use. Only a single gradient at the end of `within-region` would travel across the sheet to signal completion to the rest of the sheet. For complex shapes with many substructures, this would significantly lower the total communication. Also it would allow a data-flow like concurrency in different regions of the sheet.

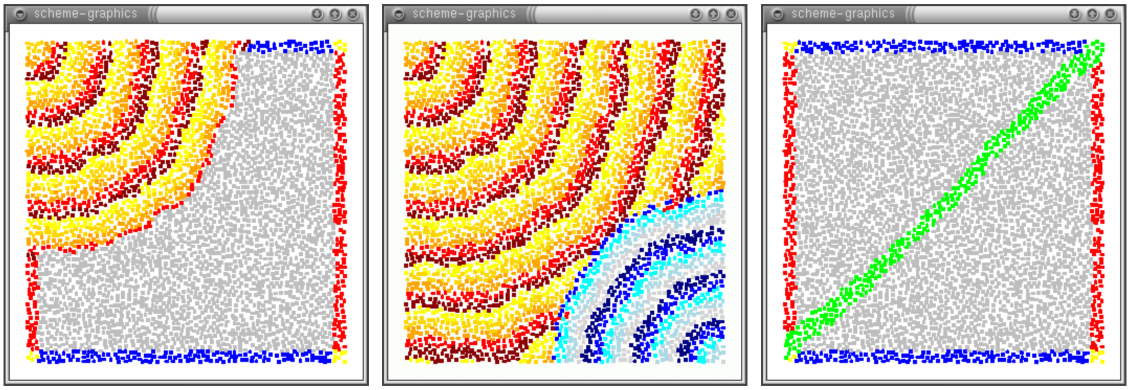
Another interesting property is that most of the cell code is conserved across all shapes. The majority of the cell code is devoted to implementing the primitives (like gradients) and the OSL operations. This is fixed across all shapes. Only a small part of the code corresponds to the actual shape sequence. This has an interesting analogy to biology as well: DNA is highly conserved across all living things and probably a large part of it is dedicated to cell functioning.

4.4 Example: Cup Revisited

To demonstrate how this works, this section presents the simulation sequence for the formation of the cup.

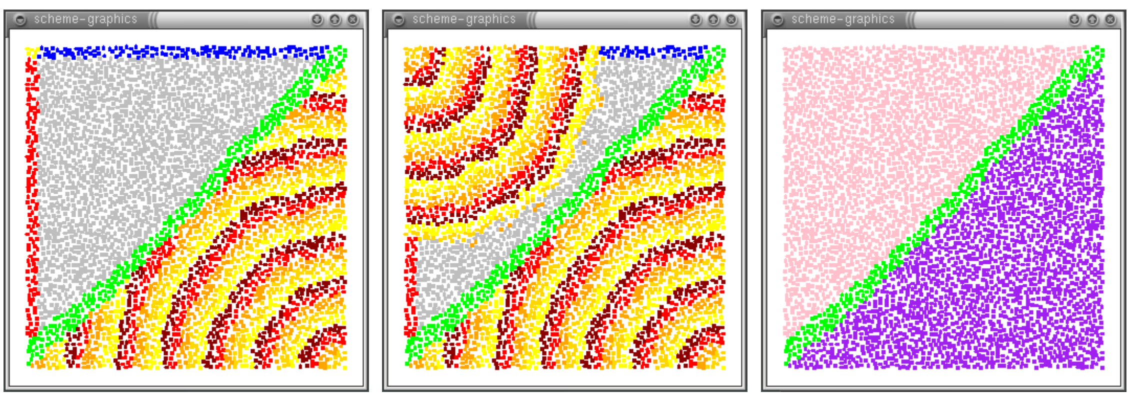


The initial state of the sheet: cells that are part of the edges are colored blue `(e12, e34)` or red `(e23, e41)` and the cells in the corners are colored yellow.



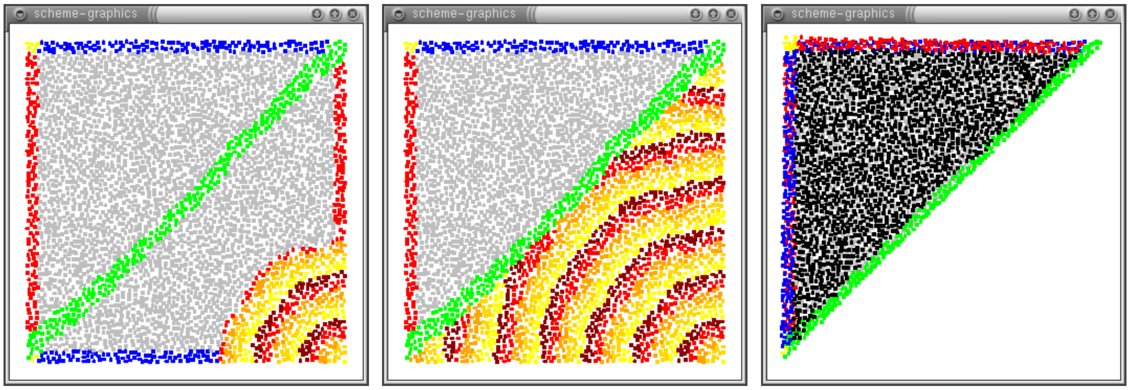
```
(define d1 (crease-p2p c1 c3 green))
```

First cells from `c1` produces gradient g1. When that reaches `c3` then the cells in `c3` produce gradient `g2`. When cells have received stable value of both gradients they compare to see if they lie along the equidistant line, and if so color themselves green and set `d1` to be true. When `g2` reaches `c1`, the `c1` cells send the final gradient `g3` marking the completion of the `d1` crease.



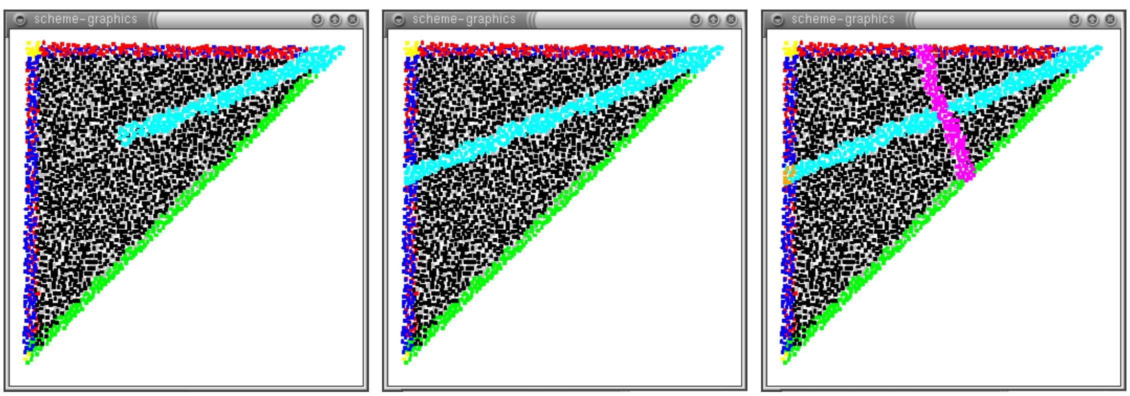
```
(define front (create-region c3 d1))
(define back  (create-region c1 d1))
```

Once the cells in `c3` see `g3`, they move on to creating the front region. The cells emit a bounded gradient `g4`. The cells along `d1` do not propagate any messages for `g4` so the gradient stops. If the line of `d1` cells were discontinuous or the cells in `c3` started the gradient before the previous axiom had completed, then the gradient would leak — one example of how things can fail. After the front region is formed the cells in `c1` form the back region in the same way.



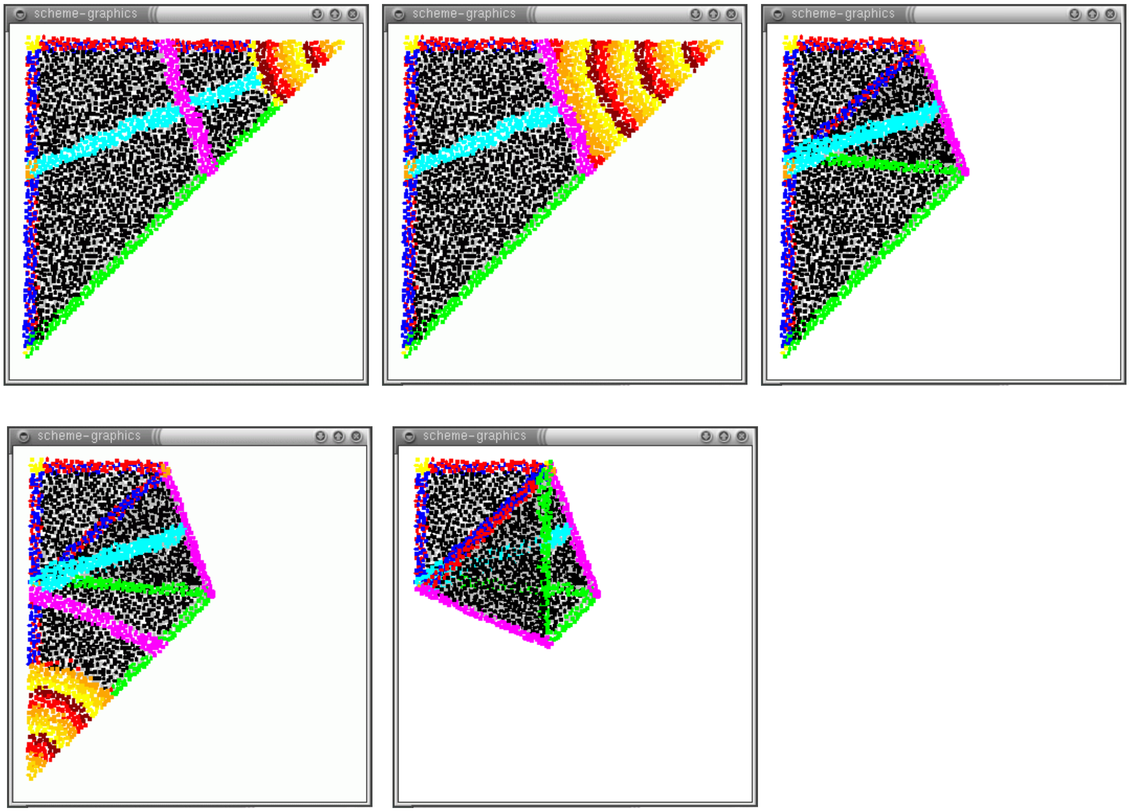
```
(execute-fold d1 apical landmark=c3)
```

Before executing a fold on `d1`, the landmark cells create a region bounded by `d1` and after the fold the cells in that region invert their polarity. Polarity is only important for executing folds. When the landmark gradient `g8` reaches the d1 cells, they start to collect neighboring values and locally estimate the fold direction. The simulator computes and performs the fold.



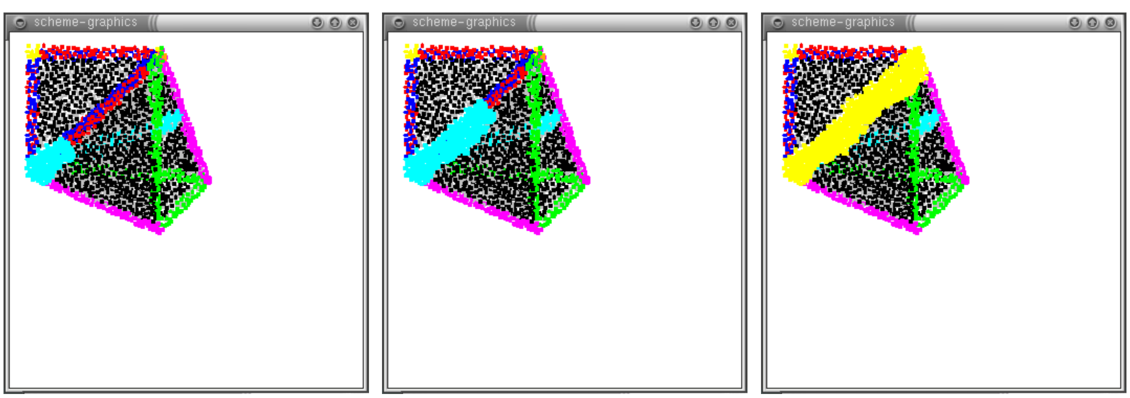
```
(define d2 (crease-l2l e23 d1 cyan))
(define p1 (intersect d2 e34))
(define d3 (crease-p2p c2 p1 magenta))
```

These pictures show the `d2` and `d3` creases forming. Because the gradients slowly travel across the sheet there is always a lag across the sheet in some direction. Nearby cells tend to remain close in time but distant cells are likely to have a time lag. As a result the `d2` crease forms gradually across the sheet in a direction related to the previous axiom. The formation of `d3` involves first finding the cells that lie on both `d2` and edge `e34`. If there are no cells at the intersection the shape can fail, therefore it is important to have sufficiently dense cells and wide creases.



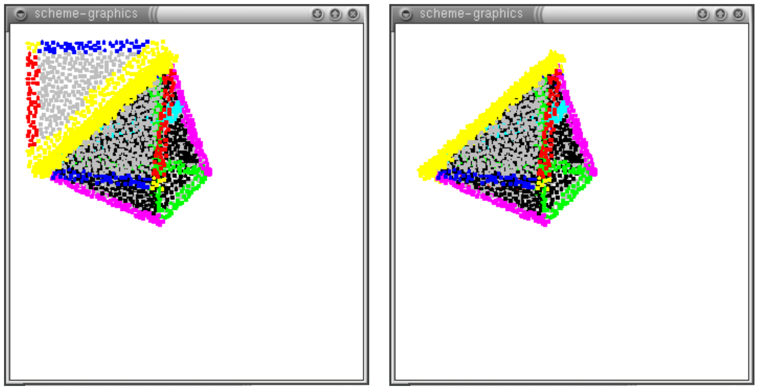
```
(execute-fold d3 apical landmark=c2)
(define p2 (intersect d3 e23))
(define d4 (crease-p2p c4 p2 magenta))
(execute-fold d4 apical landmark=c4)
```

The pictures show the formation and folding of both lateral flaps of the cup. Maintaining the apical basal surface polarity ensures that both flaps fold onto the same side



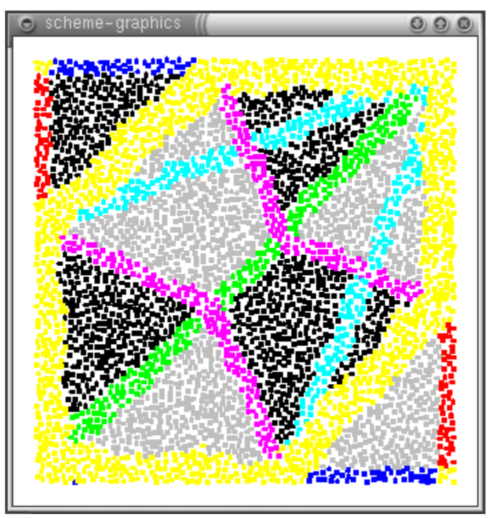
```
(define l1 (crease-lbp p1 p2 yellow))
```

This shows the `l1` crease forming using tropism. The cells in `p2` create a gradient `g7` and the cells in `p1` compete such that the cell with the lowest value starts the crease.




```
(within-region front (execute-fold l1 apical landmark=c3))
(within-region back  (execute-fold l1 basal landmark=c1))
```

The `l1` crease seeps through all layers of the sheet. In order to separately fold the front and back layers, regions are used. First only the `l1` cells with `front` variable true fold. This results in a fold of only the front layer. Then the `l1` cells that in the `back` region fold and the cup is complete



This shows the state of the cells if we were to open the sheet. This can be compared to the "crease-pattern" of our paper cup in figure 3-3. This shows how the crease lines seep through multiple layers. It also shows the eventual polarity of the cells: gray is the initial polarity, black is inverted.

The self-assembly of the cup depends critically on the axioms producing straight creases at the right places. If the crease is formed in the incorrect place, lines that should intersect may not and the layers may fold incorrectly. If the creases are discontinuous, regions may leak. All of these failures are in fact rare even though the cells are randomly distributed. This depends having sufficiently dense cells and in chapter 6 I show that an average neighborhood of 15 cells is sufficient to produce good and reliable axiom behavior and thus reliable shape formation.

The compilation process from global to local is easy to understand. However the cell programs generated are no different from any other emergent systems — the eventual shape "emerges" as a result of local interactions between the cells. The cell program by itself does not determine the final outcome. For example, the initial conditions and initial shape are implicit and can drastically change the results; a gradient from a line of cells has dramatically different effect than a gradient from a point of cells, even though the cell code is the same; the effect of actuation on the environment is not encoded in the program but significantly affects the execution. In chapter 7, I show that the initial conditions encode very important information, and the same cell program can in fact generate many related shape without modification.

Chapter 5

Shapes and Patterns using OSL

Using the Origami Shape Language one can generate many different types of shapes and patterns: flat layered shapes, all plane Euclidean construction patterns and a variety of tessellation patterns. In this chapter I present several examples in each category, that highlight different properties of the global and local languages. The axioms can be used to predict what kinds of shapes and patterns can be described by the OSL language, and hence what shapes and patterns can be self-assembled by the cells.

The Origami Shape Language provides a convenient and abstract way of generating and reasoning about shape and pattern from locally interacting cells. All of the examples in this chapter were generated by specifying the construction on a continuous sheet using OSL, compiling the OSL program to generate the cell program and then executing the cell program on the simulated sheet of identically-programmed, locally-interacting cells. The initial conditions are always the same: boundary conditions and apical/basal polarity. In the simulations the number of cells vary from 2000 to 8000 and the expected local neighborhood of a cell is between 15 to 20 cells. The cells are randomly but densely distributed. Details of the simulation environment were presented in section 2.4.

5.1 Flat Layered Shapes

Using the OSL language one can generate flat layered shapes composed of simple folds. A simple fold implies that one straight line is folded at a time and the structure lies flat after each fold (also called book folds or map folds). This is a subset of all flat layered shapes. In this section I present some examples. There is a large literature of such shapes in origami.

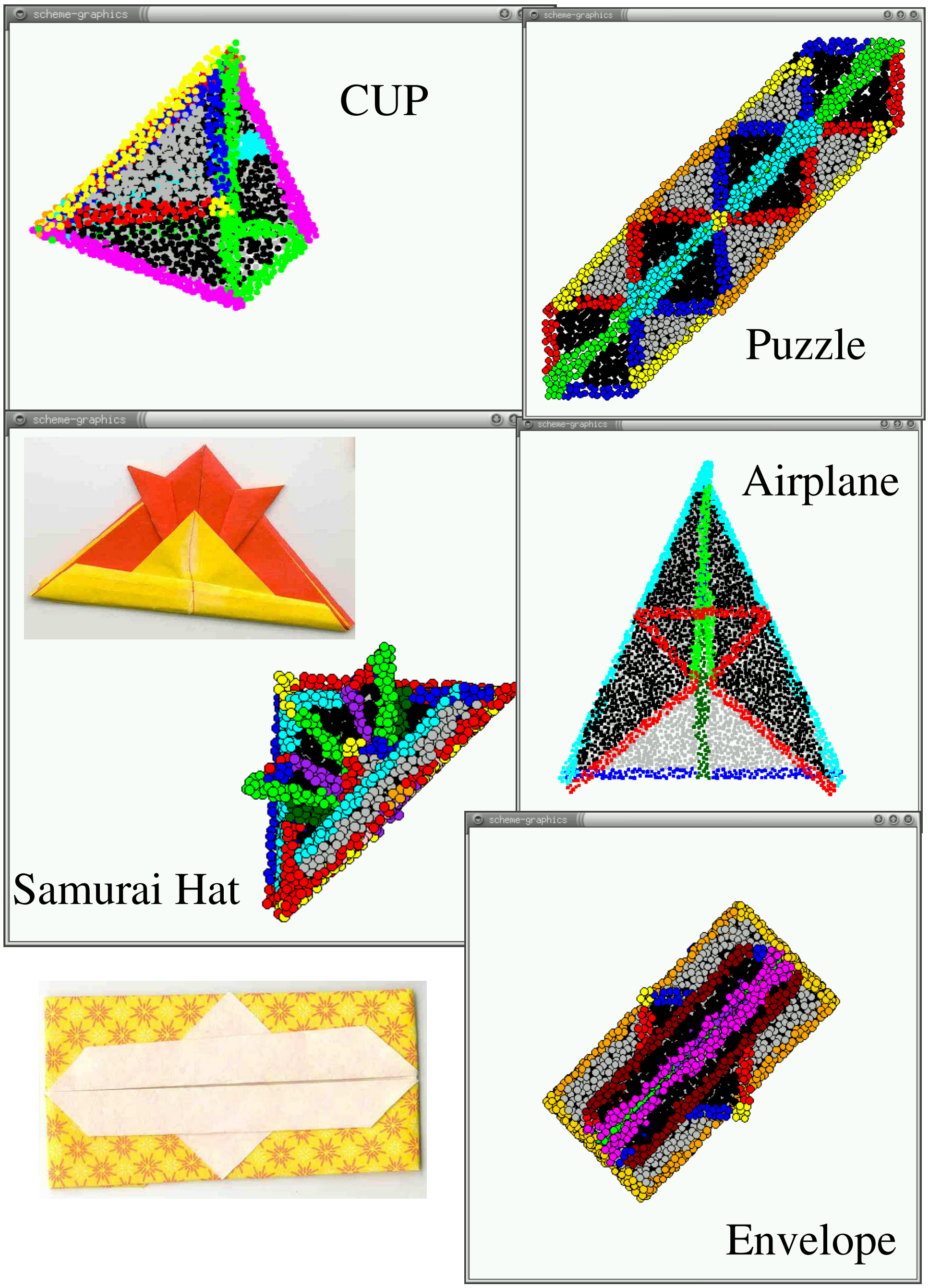


Figure 5-1: Flat layered shapes composed of simple folds

Samurai Hat

The samurai hat is a very traditional flat origami that can be created out of simple folds. It uses many partial layer folds. In forming the sides of the hat, we create two flaps, one within another using regions. The bottom of the hat is folded similar to a cup, one flap forward and one back to keep the hat together.

```
(define d1 (crease-p2p c1 c3 "green"))
(define l1 (crease-p2p c2 c4))
(execute-fold d1 apical landmark=c1)

(defun (fold-hat-side corner edge)
    (define d2 (crease-p2p corner c1 "cyan"))
    (define-region flap1 (corner d2))
    (execute-fold d2 apical landmark=corner)

    (within-region flap1
        (define d3 (crease-l2l edge d2 "yellow"))
        (define-region flap2 (corner d3))
        (execute-fold d3 apical landmark=corner)
        )
    (within-region flap2
        (define d4i (crease-l2l d1 d3 "darkgreen"))
        (define d5 (crease-l2l d1 d4i "purple"))
        (execute-fold d5 apical landmark=corner )
        )
    )

; fold right and left sides of the hat
(fold-hat-side c2 e12)
(fold-hat-side c4 e14)

; fold front and back bottom of the hat
(seepthru #f)
(define m1 (crease-p2p c1 (intersect d1 l1) "red"))
(define m2 (crease-p2p c1 (intersect m1 l1) "pink"))
(define m3 (crease-l2l m2 m1 "cyan"))

(execute-fold m3 apical landmark=c1)
(define m12 (or m1 m2))
(execute-fold m12 apical landmark=m3)
(define back-m1 (crease-p2p c3 (intersect d1 l1 "yellow") "red"))
(execute-fold back-m1 basal landmark=c3 )
```

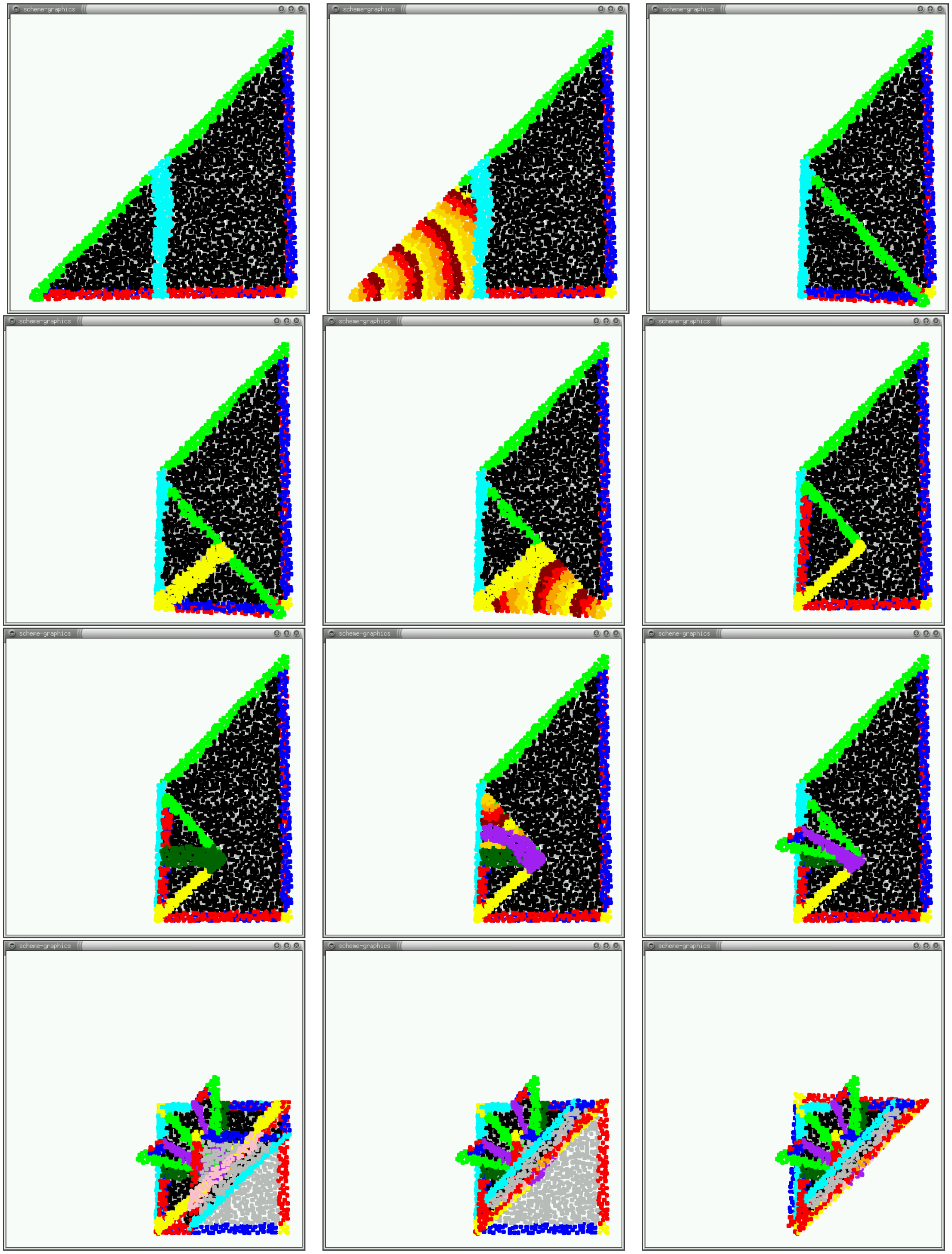


Figure 5-2: Samurai hat formation

Airplane

This code is for a common paper airplane. The symmetry of the airplane is captured using a procedure fold-wing. The regions are used to refer to two different segments of the e12 edge.

```
(define cntr-line (crease-l2l e14 e23 "darkgreen"))
(define-region rside (c2 cntr-line))
(define-region lside (c1 cntr-line))

(defun (fold-wing cntrline topline ucorner side dcorner)
    (define t2 (crease-l2l topline cntrline "green"))
    (execute-fold t2 apical landmark=ucorner)
    (define t3 (crease-l2l t2 cntrline "cyan"))
    (define p3 (intersect t2 side))
    (execute-fold t3 apical landmark=p3)
    (define t4 (crease-l2l t3 cntrline "magenta"))
    (execute-fold t4 apical landmark=dcorner)
    )

(within-region rside (fold-wing cntr-line e12 c2 e23 c3))
(within-region lside (fold-wing cntr-line e12 c1 e14 c4))
(execute-fold cntr-line basal landmark=c4)
```

The airplane points to an interesting issue regarding 3D shapes. many 3D origami shapes fold flat, and are then “opened” to create the final 3D shape. For example the last folds on the wing are opened to create roughly right angles under the wings. If the cells could calibrate how to form some simple angles then one could imagine and execute-fold command that attempted to fold some non-precise but non-flat angle. In the next section I will discuss extending these ideas to 3D shapes.

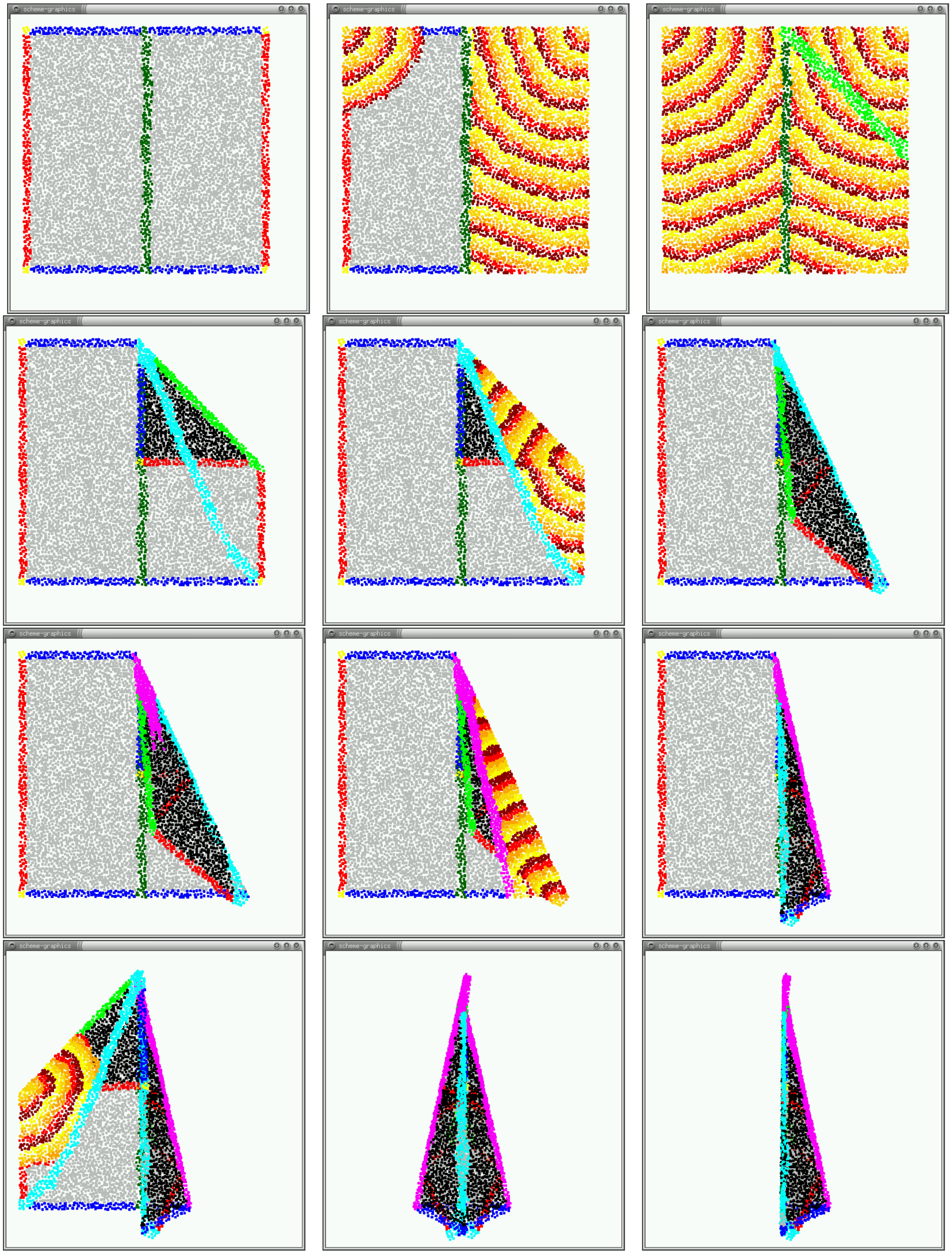


Figure 5-3: Airplane formation

Envelope

The envelope is another functional flat layered shape. The envelope requires making folds that only go through one layer. Instead of using regions, we use a different OSL primitive which allows us to turn seepthru off — meaning that cell-to-cell contact is off and each fold is as if it were performed on the flat sheet. This is a much more natural way of expressing the code for the envelope. The same code could however be expressed using regions as well.

```
(seepthru #f)
(define d1     (crease-p2p c1 c3 "green"))
(define t1     (crease-p2p c2 c4))
(define mid-pt (intersect d1 t1))

; fold a side of the envelope
(defun (envelope-side corner)
    (define d2 (crease-p2p corner mid-pt "magenta"))
    (define d3 (crease-l2l d2 d1 "orange"))
    (execute-fold d3 apical landmark=corner )

    (define p1 (intersect d3 t1))
    (define d4 (crease-p2p corner p1 "darkred"))
    (define p2 (intersect d2 t1))
    (define d5 (crease-p2p corner p2 "purple"))

    (execute-fold d4 apical landmark=corner )
    (define l (or d2 d5))
    (execute-fold l apical landmark=d4)
    ))

(envelope-side c3) ; right side
(envelope-side c1) ; left side

; fold ends to seal (here seepthru is true)
(seepthru #t)
(define t2 (crease-p2p c2 mid-pt "gold"))
(define t3 (crease-p2p c4 mid-pt "gold"))
(execute-fold t2 basal landmark=c2)
(execute-fold t3 basal landmark=c4)
```

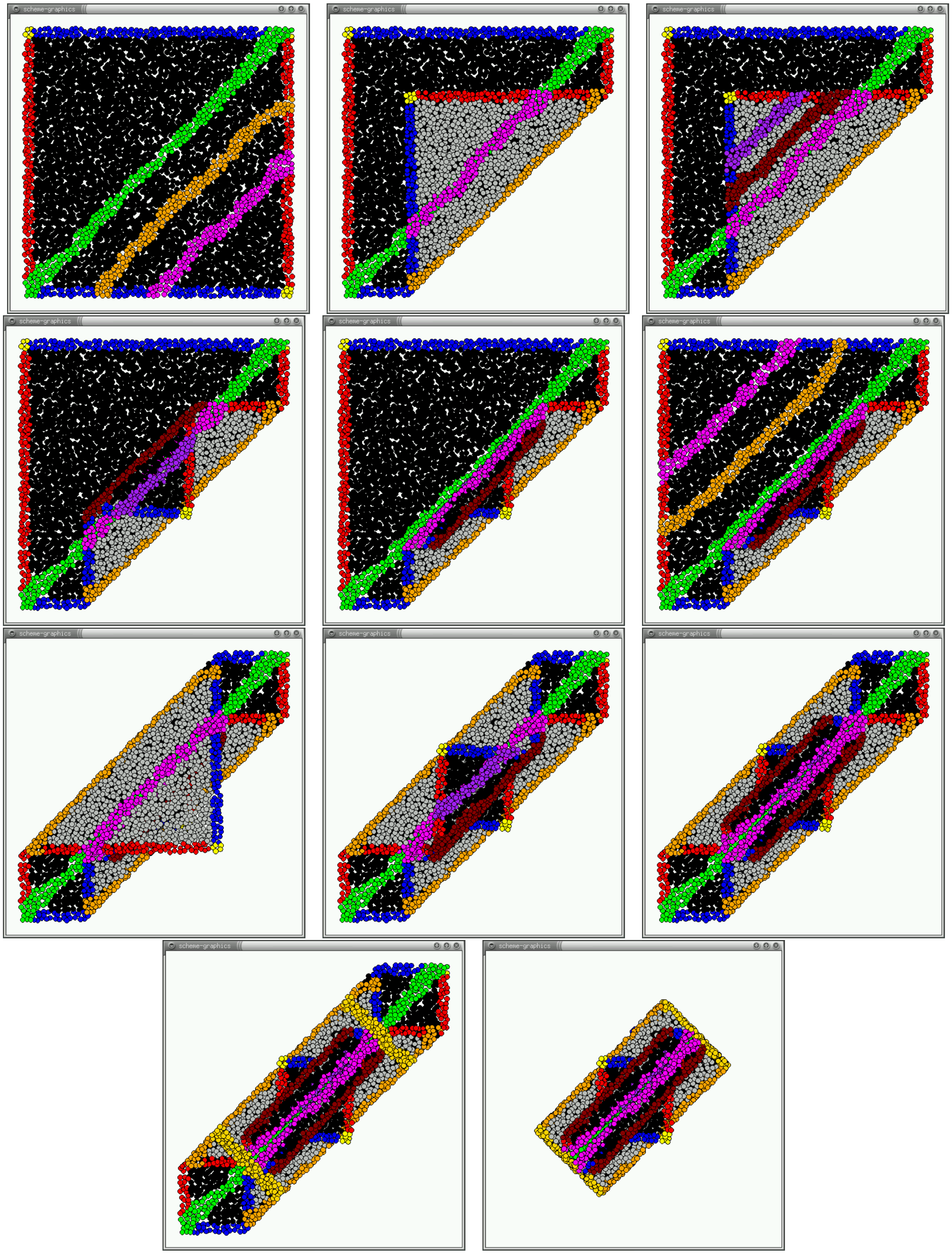


Figure 5-4: Envelope formation

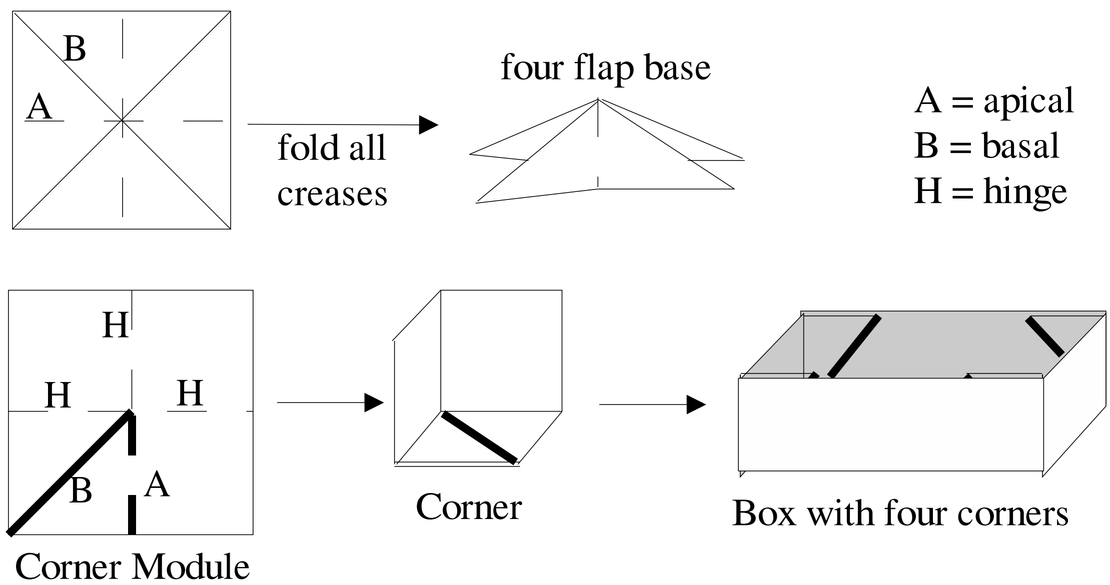


Figure 5-5: Extensions: complex base, corner module and box

Complex Folds

If several creases can be folded simultaneously then many more flat layered structures can be created. For example, a traditional origami base creates multiple flaps from a single paper by folding the crease pattern in figure 5-5. The creases must be folded together for the structure to fold flat. The crease pattern itself is generated using Huzita's axioms, therefore we can already generate the crease pattern using OSL. By extending `execute-fold` to allow multiple creases to fold at the same time, the OSL language can create this base. The creases divide the sheet into polygonal regions. In order to redetermine the apical basal surfaces, a set of polygonal regions must be chosen that will reverse their apical/basal polarity. This can be done using multiple landmarks. Using complex folds we can create structures such as the *miura-ori* fold, that is used to compactly fold and unfold paneled surfaces such as large solar panels for satellites [50].

Simulating such shapes however requires a more sophisticated underlying sheet simulator that can compute the effect of multiple folds on the sheet. Not all crease patterns fold flat. In origami mathematics, there are several theorems regarding flat foldability of crease patterns [29, 9]. Finding an abstraction between the detailed dynamic simulation and the current simple flat fold model, will allow the simulation of flat layered shapes with complex folds.

5.1.1 Extension to 3D Shapes

Currently the thesis focuses on only flat folding structures. One reason for doing so is that in the current epithelial cell model it is unclear how cells would measure the angle being created. Cell actuation is likely to be imprecise and the width of the crease varies along the length, therefore there would need to be some sort of feedback in order to create precise angles. One option would be to add some sort of angle sensing mechanism to the cells. Another option would be to use end conditions, such as contact, to determine when to stop folding. For example neural folds in embryogenesis fold until the two sides touch each other [72].

In the field of origami there has also been some work on 3D structures that relies on the creation of self-stabilizing structures. It is difficult to create precise folds with paper. However one can take advantage of the stiffness of the paper and its resistance to deformation. By precreasing along certain lines, the stiffness can be broken and the line acts like a hinge. The 3D shape is then composed of flat folds, rigid planes and hinges. An example is the corner fold in figure 5-5 where the three hinges balance each other to create the corner. These sorts of folds form molecule crease-patterns from which more complex shapes can be constructed, such as the box. There are many well known such molecules in origami [45, 41]. Many different polyhedra can be constructed from corner molecules of various types. Unlike origami, where paper cannot actively participate, the cells along the fold can actively hold a fold together or one can allow adhesion between cells (glue) to strengthen the corners of the box. Another approach used in origami is the creation of 3D structures as flat origami but then "opening" a couple of creases in the last step. For example if the creases that form the cup try to open a bit, then that opens the mouth of the cup. Several different origami boxes are produced by similar spreading of two creases at the end of the structure formation. Here the technique depends on creating a locked structure through the original flat folding sequence. In all of the cases above, the crease patterns are generated using the same set of axioms. Therefore the OSL language can generate these crease patterns. Fold execution and the notion of apical-basal surfaces would need to be made more general so that multiple surfaces can be referred too.

5.2 Plane Euclidean Constructions

This language can also be used to create patterns. Huzita proved that the origami axioms are equivalent to the Euclidean axioms. Which means that theoretically we can *self-assemble any plane Euclidean construction*, i.e. any pattern that can be described using straight edge and compass construction. Practically there are resolution limitations based on how many cells there are. However the important point is that a) we can say something about what kinds of patterns can be created by this system and relate that to general 2D geometry b) this is gives us a systematic way of self-assembling a very large class of patterns. Here are some examples of patterns in this category.

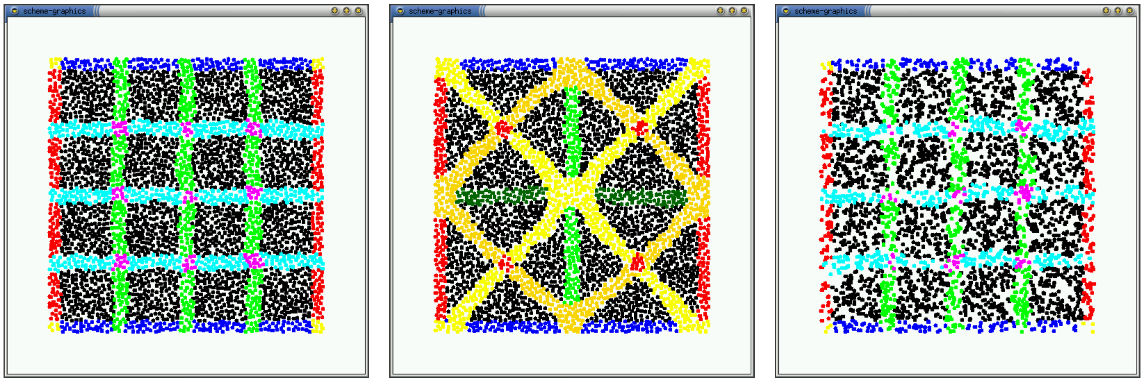


Figure 5-6: Grid and triangulation patterns on randomly distributed cells.

Grids and Triangulation

It is very simple to create regular patterns, such as the grids and triangulation in figure 5-6 using axioms 2 and 3 (crease-p2p, crease-l2l). Such patterns can be applied in many ways: 1) embedding regular structures on irregularly placed sensors for triangulation or data partitioning 2) compact folding 3) creating a programmable assembly line, i.e. programming paths on a surface without needing to provide each sensor with global coordinates. An interesting property of these regular patterns is that even when the processors completely randomly distributed (i.e. allowed to overlap arbitrarily) the patterns created are surprisingly regular. This is because axioms 2 and 3 compose gradients in a manner that minimizes error and produces geometrically accurate creases. This is discussed in detail in chapter 6.

Pattern formation is an important problem, because pattern translates to function. For example the crease pattern translates (not unambiguously) to shape, and in biology pattern formation is a fundamental piece of morphology, circulation, growth, regeneration, etc. Many applications can be thought of as a pattern formation problems; for example, how does one create the desired path on a programmable assembly line or where does one put the limbs on a reconfigurable robot.

Inverter: Topology versus Geometry

There are many interesting complex non-regular patterns that can be generated using OSL. In this case I show a caricature of a CMOS inverter pattern. This example is inspired by Coore's Growing Point Language (GPL) for organizing interconnect patterns on an amorphous computer [13]. The signature example was a set of identically programmed cells organizing to form an inverter pattern. The Origami Shape Language can also achieve the same pattern, albeit using a very different local program and with different global properties. This also alludes to one of the striking parallels between this work and GPL: both can create Euclidean constructions. However OSL focuses on geometry whereas GPL focused on topology. Figure 5-7 shows a comparison of inverter patterns generated by GPL and OSL.

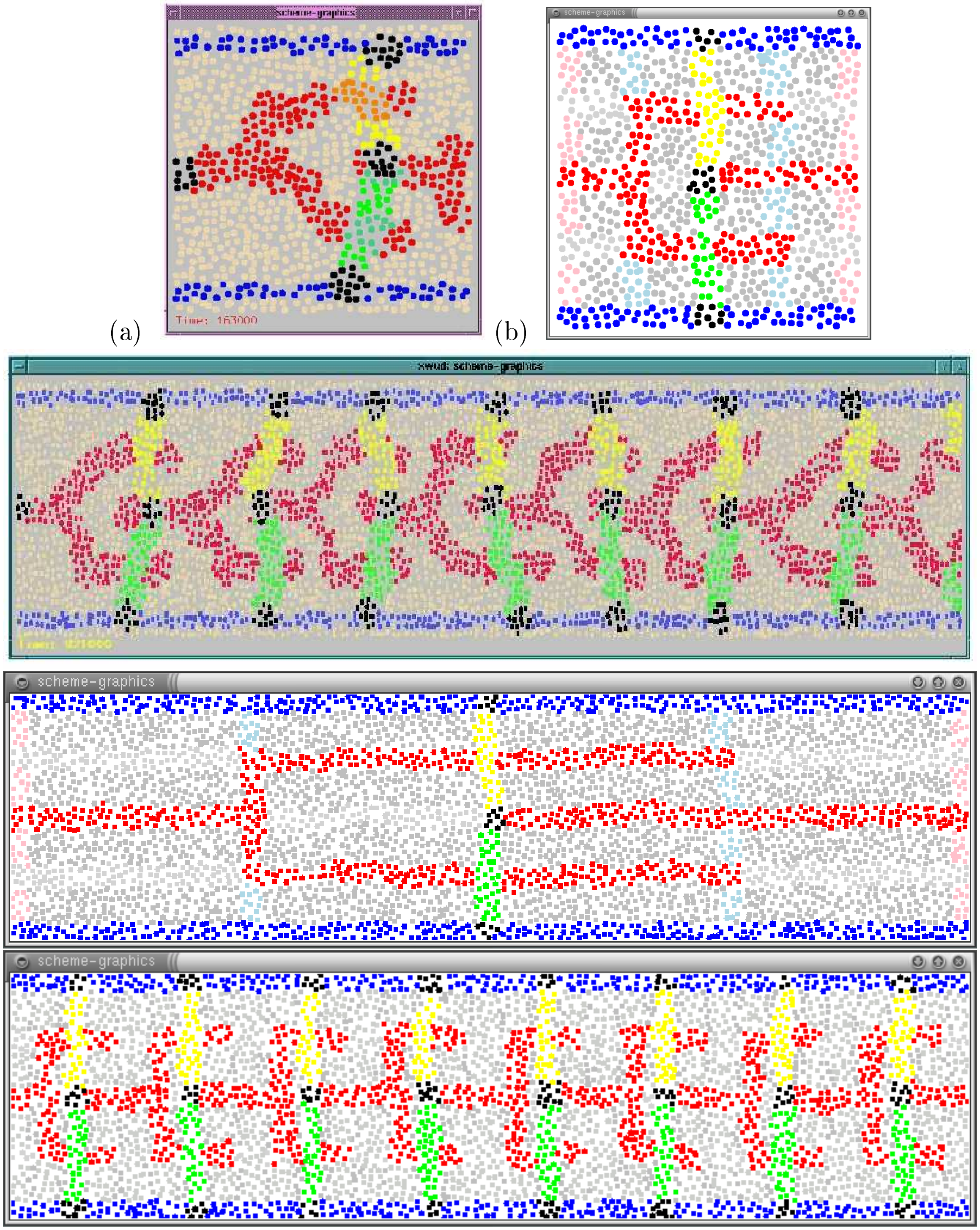


Figure 5-7: Comparison of inverter patterns generated by OSL and GPL: (a) GPL inverter (b) OSL inverter (c) the same GPL program run on a longer sheet produces a chain of inverters (d) the same OSL program produces a stretched inverter (e) an inverter chain using OSL.

```
;; OSL Program for creating an INVERTER
;;--------------------------------------

;; Creating a GRID
;;-----------------
(define h1 (crease-l2l e34 e12 "lightgray"))
(define h2 (crease-l2l h1 e12 "lightgray"))
(define h3 (crease-l2l h1 e34 "lightgray"))
(define v1 (crease-l2l e23 e14 "lightblue"))
(define v2 (crease-l2l v1 e23 "lightblue"))
(define v3 (crease-l2l v1 e14 "lightblue"))
;; Vertical regions
(define IN  (create-region IN  (e14 v3))
(define OUT (create-region OUT (e23 v1))
(define tmp1 (or v2 v3))
; MID = a

 region bounded by two creases
(define MID (create-region MID (v1 tmp1))
;; Horizontal Regions
(define UP   (create-region UP   (e12 h1))
(define DOWN (create-region DOWN (e34 h1))
(define tmp2 (or h2 h3))
(define CNTR (create-region CNTR (h1 tmp2))

;; Actual laying down of Material
;;-------------------------------
;; POWER
(color (or e12 e34) "blue")
;; POLY
(within-region IN   (color h1 "red"))
(within-region MID  (color (or h2 h3) "red"))
(within-region OUT  (color h1 "red"))
(within-region CNTR (color v3 "red"))
;; METAL
(within-region UP   (color v1 "yellow"))
(within-region DOWN (color v1 "green"))
;; CONTACTS
(define contacts (intersect v1 (or e12 h1 e34)))
(color contacts "black")
```

Figure 5-8: OSL program for an inverter pattern

The GPL inverter starts by initial seed location growing a line parallel to two predefined power lines (cells in blue). The line then splits into two branches, which grow towards the rails and then parallel to the rails and eventually sprout metal lines. At the cell level, all lines are generated by growing creases towards and away from multiple gradients. Axiom 1 is based on a very simplified version of this idea, but essentially has the same behavior. The lines grow from point to point and therefore guarantee connectivity between points. However local variations in the density of cells causes the lines to wiggle around.

The OSL inverter on the other hand is generated by subdividing the sheet into regions and then laying down specific lines within the regions (code in figure 5-8). One can think of this abstractly as generating the grid in figure 5-6 and then drawing the inverter on top of that. The lines of the inverter are generated mainly by using axioms 2 and 3. In both axioms 2 and 3 multiple gradient values are compared autonomously by cells to determine whether or not they lie on the crease line. This method generates very visually straight creases and is not as affected by local variations in density. However these axioms do not guarantee connectivity; each cell autonomously decides whether or not it is part of the crease.

Figure 5-8 shows the OSL code for the inverter pattern. The sheet is first divided into a grid, and the grid is used to divide the sheet into horizontal and vertical regions (`IN,OUT,MID` and `UP,DOWN,CNTR`). Then we can talk about segments of lines, such as the line `h1` within the region `IN` and assign those segments special state. Notice that again the program is abstractly at the level of lines; cells and gradients do not have to be considered in order to create complex patterns.

Because of the different ways in which the inverters are formed, both the GPL and OSL inverters react differently when run on longer sheets. The GPL program encodes a specific length for inverter parts, and if there is more area then additional inverters are formed. By contrast the OSL inverter pattern simply *stretches* with the area of the region (figure 5-7). This is because the pattern itself was created relative to the boundary of the sheet.

An inverter chain can also be created in OSL in a modular fashion (figure 5-7). The single inverter OSL program is written as a procedure that takes the left and right border as an argument. Then the overall sheet is divided into segments and the procedure is called within each segment. The inverters can be created one at a time, or inverters in disjoint regions can be created *concurrently*. This is because the disjoint regions isolate the formations and allow the same axioms to be executing in multiple places without interference. Doing this however violates some level of abstraction — at the global level we are aware of potential interference problems between inverters. The OSL program however is still written without any reference to cells or gradients. The following is part of the OSL code for creating the inverter chain. I have omitted the code for subdividing the sheet into 8 segments, which is tedious but straightforward. This code was compiled to generate the cell program run on all of the cells. Figure 5-9 shows the result. The inverter chain forms by first segmenting the region into eight segments and then the inverters in the alternate regions form concurrently.

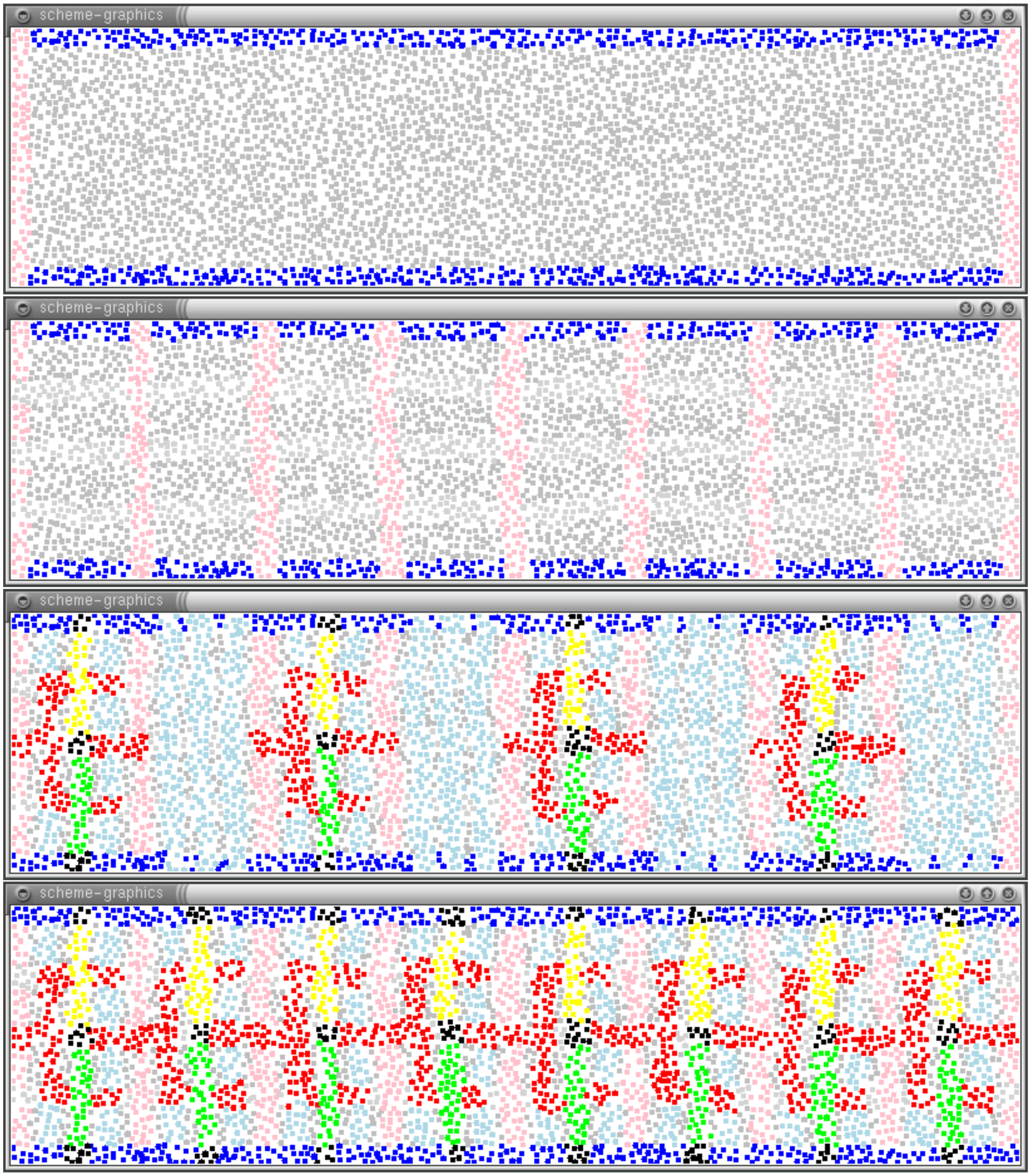


Figure 5-9: Formation of an inverter chain. The inverters in odd segments are created concurrently, followed by the inverters in the even segments. Using disjoint regions prevents interference between the concurrent formation of the inverters.

```
;; Omitting code for forming the following segments....
;;  -----------------------------------------
;;  | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 |  (segment names)
;;  -----------------------------------------
;; e14  s1   s2   s3   s4   s5   s6   s7   e23 (segment borders)

; Defining a collection of disjoint (odd) regions
(define region-odd  (or r1 r3 r5 r7))
(define oleft       (or e14 s2 s4 s6))
(define oright      (or s1 s3 s5 s7))
; Creating the odd inverters
(within-region region-odd (draw-inverter oleft oright))

; Defining a collection of disjoint (even) regions
(define region-even (or r2 r4 r6 r8))
(define eleft       (or s1 s3 s5 s7))
(define eright      (or s2 s4 s6 e23))
; Creating the even inverters
(within-region region-even (draw-inverter eleft eright))
```

5.3 Tessellation Patterns

The language can also be used to create a variety of tessellation patterns by first folding the sheet into a square or triangular tile, making patterns on the tile that seep through, and then unfolding the sheet. Figure 5-10 shows two examples. In the first example the pattern is created by folding the sheet into a small triangular tile and creating a pattern on that tile. The second example shows a cup crease pattern formed by first folding the sheet into a square of one quarter size and then creating a cup on that smaller square. The tile patterns are mirror images across the connected tile edges. This form of creating patterns is a similar idea to cutting out snow flakes from a sheet of paper. However it does require that the OSL language be extended to allow the sheet to unfold. For tessellation patterns this is a simple extension because the unfolding sequence is the exact reverse of the folding sequence. Therefore each unfold is a simple unfold (i.e. the structure lies flat after each unfold). The cells could locally determine the crease direction and loosen the fibers along that direction. The extension to unfold has not been implemented yet and in each of the simulations the sheet was unfolded globally by the simulator.

The advantage of creating patterns this way is that it ensures symmetry in the pattern across the fold line. Instead of creating each repeated pattern independently, the pattern seeps through the cells and thus the images in across the fold-line match well. It is interesting question whether symmetric patterns such as butterfly wings take advantage of folding.

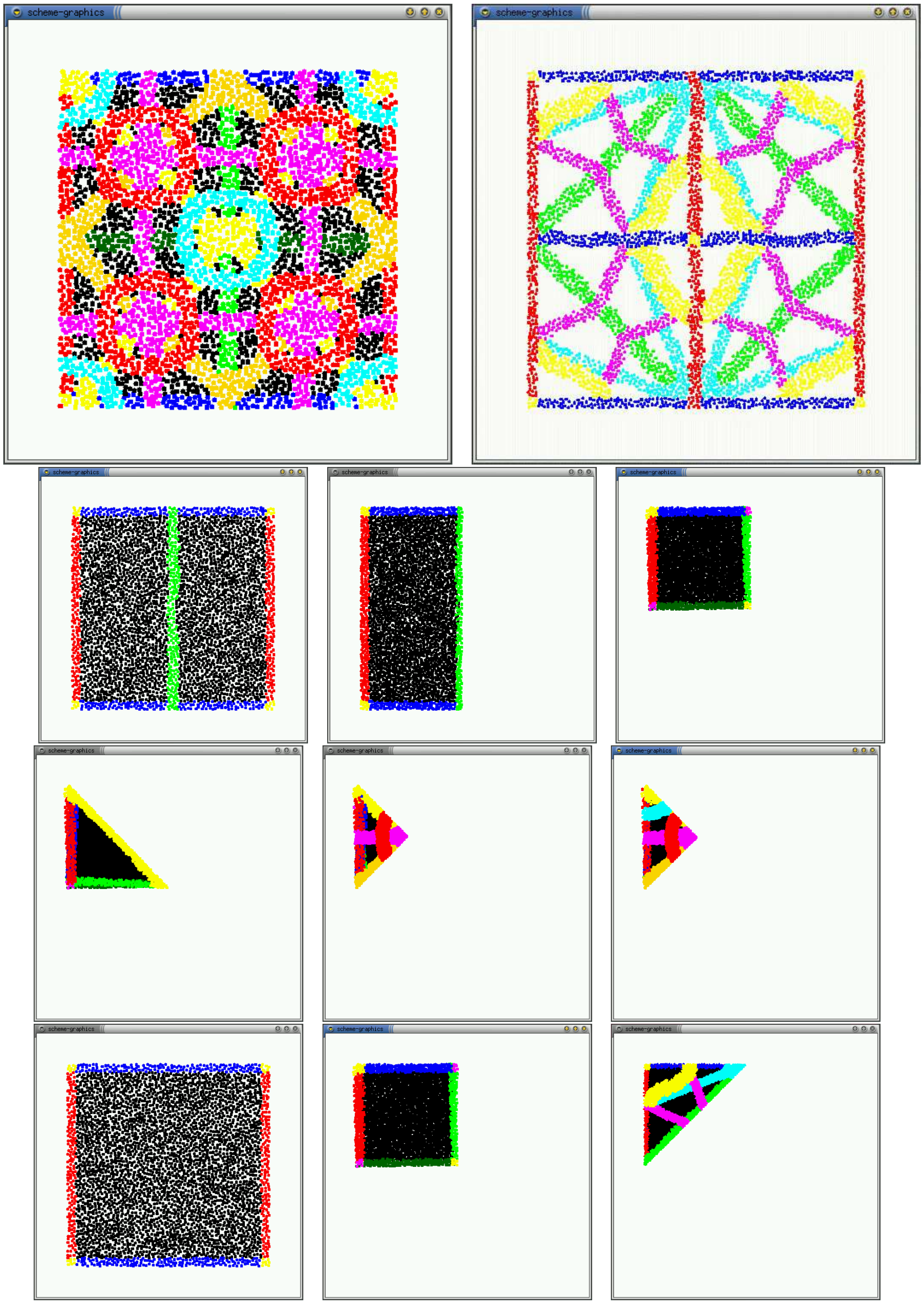


Figure 5-10: Two tessellation patterns. The second tessellation is created from the cup crease pattern.

Chapter 6

Achieving Robustness: Theoretical and Experimental Results

Achieving robustness without relying on regular grids, access to global information, or perfectly reliable individuals is an important focus of this work. The motivation comes from the desire to build applications such as programmable materials and intelligent environments that embed the computation of millions of smart sensors and actuators into surfaces, materials, and the environment. To cheaply bulk manufacture billions of elements will require eschewing assumptions such as unique identifiers and perfect reliability. It is also necessary to avoid assumptions of precise positioning such as regular grids if these elements are to be embedded into materials or sprinkled onto surfaces. Depending on centralized global knowledge, such as global clocks or external beacons for triangulating position, puts limits on the applications and environments and is inherently vulnerable to catastrophic failure. The same can be said for algorithms that assume particular interconnect topologies — the algorithms are then vulnerable to simple disruptions in the grid. Even so, such assumptions are common in current research [47, 74, 8].

Developmental biology suggests that it should be possible to construct complex structures reliably without central control and without stringent global assumptions. My work uses inspirations from biology to develop primitives that discover necessary information on randomly distributed unreliable cells without global clocking, global identifiers or access to global coordinates. Instead, robustness is achieved by depending on large and dense populations, using average behavior rather than individual behavior, trading off precision for reliability, and by avoiding any centralized control.

In this chapter I present theoretical and experimental analysis to show that an expected local neighborhood of 15–20 cells is sufficient to reliably control the self-assembly of shapes and geometric patterns on randomly distributed cells. The point of this chapter is to demonstrate that not only is it possible to achieve reliable and predictable complex behavior without stringent global assumptions, but it is also possible to "analyze" emergent behavior.

6.1 Robustness at a Glance

The previous two chapters presented many simulations of shapes formed using the Origami Shape Language. These formations could fail in many possible ways: cells forming an incorrect or crooked crease line, no cells at the intersection of two creases, errors in gradients and cell-to-cell contact, leaking regions etc. However such failures are extremely rare and the simulations presented are very robust. The following is a brief summary of the mechanisms by which this robustness is achieved.

1. Points and creases are always represented by groups of cells, all of which are equal. Thus the behavior of a point or line is the average behavior of many cells and the equality of the cells provides robustness against individual cell death.

2. Gradients produce good (but not error-free) estimates of distance on random distributions of sufficient density. The process by which a gradient is created is robust to random cell death and occasional message loss. Gradients also tend to average out small variations and thus errors by single cells, failures in cell-to-cell contact, and variations in the shape of points and width of creases tend not to have a significant effect.

3. The axioms do not rely on perfect distance estimates, instead they use local comparisons of gradient values and use many short-lived gradients to achieve error tolerance. Each axiom creates new gradients to minimize error specifically in the area where the crease is expected. Gradient values are not reused, so errors in a single gradient have limited effect. Each axiom also attempts to produce creases with sufficient cell width, independent of the number of cells in the inputs. Hence imperfections in creases tend not to propagate.

4. The overall program achieves robustness by avoiding centralized and hierarchical control. Control constantly shifts around the sheet and new centers of control are constantly being created.

5. All of the above depend on a large and dense population of cells. This is necessary for achieving good gradient behavior on random distributions and high probability of sufficient cells in points and creases. The random distribution of cells also discourages algorithms that depend on the precise position of individual cells. This automatically provides some robustness against cell death; even if a few cells die, the interconnect is still essentially the same.

Together these mechanisms make the individual operations of OSL extremely reliable and reasonably precise. For example the likelihood that a crease does not form at all is extremely small and the probability that the crease will be a reasonable approximation to the desired line is high. The shape formation depends critically on the axioms ability to reliably form straight and accurate creases. In this chapter I provide an extensive analysis of the robustness of the axioms. However still are some limitations to the robustness of the system.

1. It is vulnerable to large regional failures or holes. For example if all the cells in a point are killed, or if there is a large hole of cells where the intersection of two creases should have been, then the structure can fail to form.

2. It is vulnerable to malicious cells. If a remote cell decides that it is a member of some point and starts to produce gradients, it can affect the entire sheet.

3. A shape formation can not recover from the complete failure of a global operation. The mechanisms are aimed at making the probability of this event extremely low, however there is no high level recovery process.

It may be possible to counteract 1 and 3 by adding some sort of control at the global level to determine whether a crease or point formed successfully, and to suggest alternate constructions. Protecting against malicious cells is more difficult because cells have no way of distinguishing information from good and bad cells. To some extent checks can be added within an individual cell to cause it to shut itself down if it seems to be on the wrong path - for example if none of its local neighbors belong to the same point. The global operations can already tolerate some fraction of cell death. Addressing these limitations is left to future work.

6.2 Random Distribution of Cells

As mentioned before, the formation of the shape depends critically on the ability of the axioms to reliably create straight and accurate creases. In this section I present analysis and experiments to show that the axioms are both reliable and reasonably accurate on randomly distributed cells, when the expected local neighborhood is greater than 15. This stems mainly from the behavior of gradients, that produce low errors in distance estimates for local neighborhoods over 15. The behavior of each axiom also depends on the inputs and I provide analysis and experiments to show how.

In this section I provide an analysis of the gradients, followed by an analysis of the axioms. For the purpose of analysis, the cells are assumed to be distributed independently and randomly on a unit square plane. This means that for each cell we choose a random x coordinate and random y coordinate on the unit square, independently of all other cells. The probability that there are k cells in a given area a can be described by a Poisson distribution [66].[1]

$$Pr(\text{k cells in area a}) = \frac{(\rho a)^k}{k!} e^{-\rho a}$$

[1] To intuitively understand why this is a Poisson distribution, think of having two buckets of size a and $S-a$, where S is the total area. If the cells are distributed independently and randomly, then the probability of a cell landing on any point on the surface S is the same. Thus the probability of landing in bucket a is proportional to the area a. This is a binomial distribution. When S is large compared to a this can be approximated by a Poisson distribution. This fact is often used in packet radio analysis [39].

From this formula, we can derive the expected number of cells in area a to be ρa. ρ is equal to $\frac{N}{S}$ where N is the total number of cells and S is the total surface area (in this case $S = 1$). The value that we are interested in is the expected number of cells in a local neighborhood. A cell communicates with all other cells within the communication radius r. Thus the expected local neighborhood n_{local} is $\rho\pi r^2$. In the actual cell sheets, the cells are randomly distributed but do not arbitrarily overlap (section 2.4), which reduces the variations in density. This random distribution represents a worst case analysis where cells may overlap arbitrarily.

6.2.1 Analysis of Gradient Robustness

The simulations in chapter 5 show that even though the cells are placed randomly, the creases formed look surprisingly straight. Furthermore the straightness seems to be independent of the orientation of the crease. The accuracy of the creases fundamentally depends on gradients providing reasonable estimates of distance. This section presents analysis that shows what local cell density is necessary for achieving good gradient behavior, and how much error to expect in the gradient distance estimate.

Integral Distance Estimate

A gradient is essentially equivalent to a breath-first-search tree [44]. It computes the shortest communication path from the source to any cell. However, due to the spatial locality of communication, the gradient also reflects the distance between the source and any cell. Let the gradient value of cell i be h_i, then the distance between cell i and the source is greater than or equal to $h_i \times r$, where r is the communication radius. In the ideal case they are equal, which would imply that with each communication hop one moved a distance r closer to the source. However given any two cells, there may not be enough intermediate nodes for the shortest communication path to lie along the straight-line path between the source and destination. In that case, the gradient value overestimates the actual distance between the cell and the source. Intuitively this is related to the density of cells within a local neighborhood.

This phenomena has been extensively studied in the context of random plane graphs and packet radio networks, which share a similar model to an amorphous computer. Receivers are spatially distributed (usually randomly) and each receiver communicates via broadcast with all neighbors within a fixed radius. The goal is usually to guarantee connectivity and optimize network throughput. Shivendra *et al.* showed that the theoretical expected local neighborhood n_{local} to ensure connectedness is between 2.195 and 10.526 and simulation experiments suggest at least 5 [59]. Silvester and Kleinrock proved that $n_{local} = 6$ produces optimal network throughput for randomly distributed receivers [39]. In the process they derived a formula for how the expected distance covered in one communication hop is affected by the parameters of the random distribution. The expected distance covered per communication hop, d_{hop}, is the physical distance between a pair of cells divided by the expected number of hops in the shortest communication path. Kleinrock and Silvester [39] showed that d_{hop} depends only on the expected local neighborhood n_{local}, not the total number of

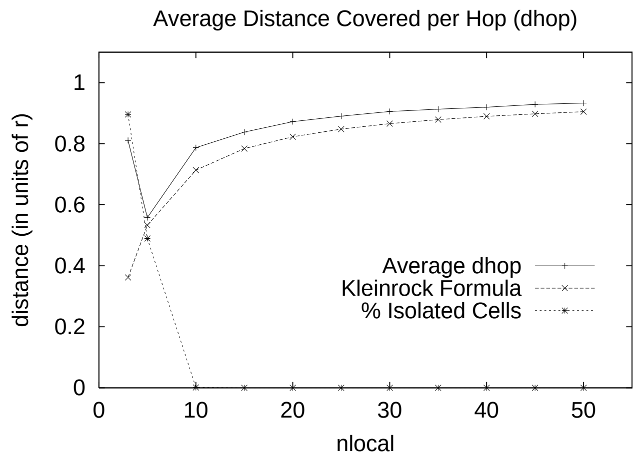


Figure 6-1: Theoretical and experimental values for the average distance covered in one communication hop d_{hop}, for different expected local neighborhoods n_{local}. There is significant improvement below $n_{local} = 15$, after which increasing the neighborhood size has diminishing returns.

cells.[2]

$$d_{hop} = r(1 + e^{-n_{local}} - \int_{-1}^{1} e^{-\frac{n_{local}}{\pi}(\arccos t - t\sqrt{1-t^2})} dt) \tag{6.1}$$

In figure 6-1, d_{hop} is plotted for different n_{local} using this formula. From this graph we can see that when the expected number of local neighbors is small, the distance covered per communication hop is small and the percentage of disconnected cells is large. But as the expected local neighborhood increases, the probability of nodes along the straight-line path increases rapidly until $n_{local} = 15$, when further increases in local cell density has diminishing returns. Hence the analysis suggests n_{local} of 15 to be a critical threshold for achieving low errors in the distance estimates.

However, even in the ideal case of infinite density, the distance estimates produced are still integral multiples of the communication distance r. This low resolution adds an average error of approximately 0.5 r to the distance estimates. Therefore we expect the error to asymptote around 0.5 r.

Experiment: In figure 6-1, I have plotted a measured value of the average distance covered per hop for different n_{local}, averaged over several simulations of a gradient from a random source. I have also plotted the percentage of unconnected cells. The result

[2]Since n_{local} is proportional to N/S where N is the total number of cells, it would seem odd to say that the formula does not depend on the total number of cells. However if n_{local} is kept constant and N is increased (which implies the total area S must increase), then N has no effect. Hence it is appropriate to say that d_{hop} depends on only n_{local}.

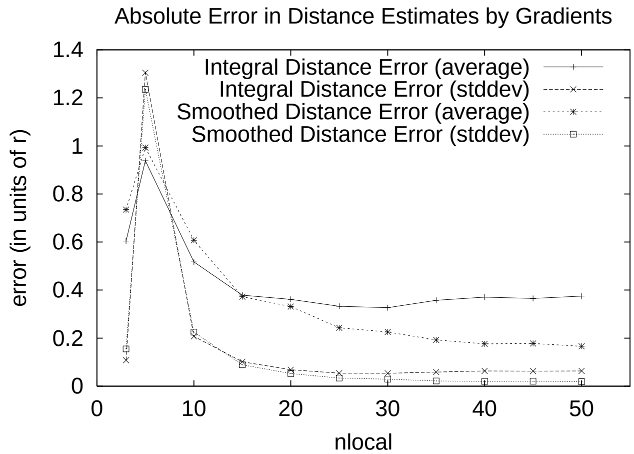


Figure 6-2: Average error in gradient distance estimates for different n_{local}. Significant improvements are seen in the integral distance estimates for $n_{local} < 15$. Beyond 15 there is improvement when the distance estimates are smoothed.

confirms that the average distance covered per hop does vary as predicted by Kleinrock and Silvester. The formula slightly under-predicts d_{hop} due to an approximation made in the proof when the source and destination are close. Also, the simulation results suggest n_{local} of at least 10 is necessary to significantly reduce the probability of isolated cells.

Figure 6-2 shows results from simulation experiments that calculate the average absolute error in the integral distance estimates for different values of n_{local}. To vary n_{local}, the total number of cells N is changed while keeping S and r constant. This keeps the physical diameter of the sheet (in units of r) constant across all simulations, so that all experiments are equally affected by any errors correlated with distance. In each simulation a gradient is produced by a randomly chosen cell in the lower left corner. The data point for each value of n_{local} is averaged over 10 simulations. The absolute error for a cell i is computed as $error_i = h_i d_{hop} - d_i$, where h_i is the gradient value, d_i is the Euclidean distance between cell i and the source, and d_{hop} is the expected distance covered per hop calculated using formula 6.1. This takes into account the fact that d_{hop} represents the expected width of a band of similar values in the gradient and computes the residual error.

The results confirm our earlier analysis. As the value of n_{local} increases the accuracy of the distance estimate improves, with both the average and standard deviations in error decreasing dramatically. However past $n_{local} = 15$ the error asymptotes at 0.4r due to the limited resolution. Further analysis of these simulations shows that the error does not increase significantly with distance from the source because the majority of the per hop error is removed by using Kleinrock and Silvester's formula (6.1). The error is also not correlated with orientation about the source which is an interesting side-effect of choosing a random distribution versus a rectangular or

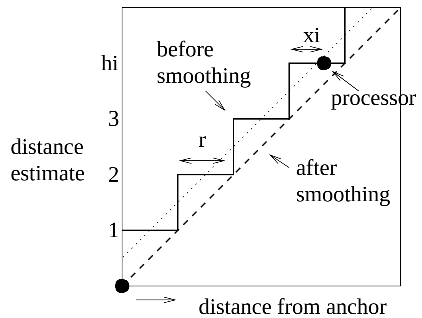


Figure 6-3: Smoothing on a 1D array of regularly spaced cells. The gradient source is in the left corner.

hexagonal grid where there is anisotropy.

Improving the Distance Estimate through Smoothing

The gradient distance estimate is improved by using local averaging. Each cell collects its neighboring gradient values and computes an average of itself and neighbor values.

$$s_i = \frac{\sum_{j \in nbrs(i)} h_j + h_i}{|nbrs(i)| + 1} - 0.5 \tag{6.2}$$

where h_i is the gradient value at cell i or in other words the integral distance estimate of cell i from the source in units of the communication radius r. $nbrs(i)$ are all the cells within the communication radius r of cell i. The formula is derived from the effect of smoothing a gradient on a linear array of evenly spaced cells.

Claim: *On a linear array of evenly distributed cells, where each cell knows its integral distance h from the left end, a cell can compute its distance by averaging the integral distances of all cells within the communication radius r, using formula 6.2.*

Proof: If we plot the value of the gradient against the position of the cell in the array, we get a staircase-like plot where the width of each step is r and the height of each step is 1 (figure 6-3). Averaging self and neighbor values produces a smooth line through the staircase which when shifted gives the correct distance.

More formally, for a cell i the integral distance $h_i = \lfloor \frac{d_i}{r} \rfloor$ where d_i is the Euclidean distance between the cell and the left end of the array. In other words $d_i = (h_i - 1)r + x_i$ where x_i is less than r. The neighborhood of cell i is all cells within the range $d_i - r$ to $d_i + r$. This region can be divided into can be divided into three parts: the cells in region $d_i - r$ to $d_i - r + x_i$ that have integral values of $h_i - 1$, the cells in region $d_i - r + x_i$ to $d_i + r - x_i$ that have integral values h_i and the cells in region $d_i + r - x_i$ to $d_i + r$ that have integral values of $h_i + 1$. Let ρ is the density of cells per unit distance, then the smoothed value at cell i is.

Using formula 6.2, the smoothed distance estimate is:

$$
\begin{aligned}
s_i &= \frac{\sum_{j \in nbrs(i)} h_j + h_i}{|nbrs(i)| + 1} - 0.5 && (6.3) \\
&= \frac{(h_i - 1)(r - x_i)\rho + h_i r \rho + (h_i + 1)x_i \rho}{2r\rho} - 0.5 && (6.4) \\
&= h_i - 1 + \frac{x_i}{r} = \frac{d_i}{r} \quad \text{q.e.d.} && (6.5)
\end{aligned}
$$

The analysis on a linear array does not hold in two dimensions because the communication region is circular and hence the proportion of cells with lower and higher integral distances is not the same as in the linear case. This would suggest weighing the neighbors differently depending on their distance from the cell. However in the absence of any positional information about the neighbors, a cell is forced to weigh all its neighbors equally. More importantly the cells are not evenly spaced and there are variations in density even within a neighborhood. Hence we expect the variations in density to add further error to the smoothing formula.

Experiment: For each of the experiments done for integral gradient values, I also calculated the error in the smoothed gradient value for each cell. The average error results are plotted in the graph in figure 6-2. The simulation experiments show that for $n_{local} > 15$ smoothing does significantly reduce the average error in the gradient value. Before that the error is dominated by the integral distance error. At $n_{local} = 40$ the average error is as low as 0.2 r. However the error is never reduced to zero due to the linear array approximation and uneven distribution of cells.

Resolution Limit

There is, in fact, a fundamental limit to the accuracy of the gradients, or for that matter any estimate developed strictly from the topology of the cell graph. We can think of each cell as a node in a graph, such that two nodes are connected by an edge if and only if the cells can communicate in one hop, i.e. they are less than r distance apart. It is possible to physically move a cell a non-zero distance without changing the set of cells it communicates with, and thus without changing any position estimate that is based strictly on communication. The old and new locations of the cell are indistinguishable from the point of view of the gradient. The average distance a cell can move without changing the connectivity of the cell graph gives a lower bound on the expected resolution achievable by the gradient.

Theorem 1: *The expected distance a cell can move without changing the connectivity of the cell graph on an amorphous computer is* $(\frac{\pi}{4n_{local}})r$.

Proof:[3]. Let Z be a continuous random variable representing the maximum distance a cell p can be moved without changing the neighborhood. The probability

[3] proof courtesy of Chris Lass, a former group member

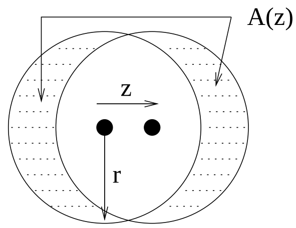


Figure 6-4: A cell can move a distance z without changing the connectivity if there are no cells in the shaded area.

that Z is less than some real value z is:

$$F(z) = Pr(Z \leq z) = 1 - e^{-\rho A(z)}$$

which is the probability that there is at least one cell in the shaded area $A(z)$ (figure 6-4). The area $A(z)$ can be approximated as $4rz$ when z is small compared to r and we expect z to be small for reasonable densities of cells. The expected value of Z is:

$$\begin{aligned} E(Z) &= \int_0^\infty z\dot{F}(z)dz && (6.6) \\ &= \int_0^\infty \rho 4rze^{-\rho 4rz}dz && (6.7) \\ &= -ze^{-\rho 4rz}\Big|_0^\infty + (-\frac{1}{\rho 4r})e^{-\rho 4rz}\Big|_0^\infty \text{ (by the product rule)} && (6.8) \\ &= -(z + \frac{1}{\rho 4r})e^{-\rho 4rz}\Big|_0^\infty && (6.9) \\ &= r(\frac{\pi}{4n_{local}}) \text{ q.e.d} && (6.10) \end{aligned}$$

Hence, we do not expect to achieve resolutions smaller than $\frac{\pi}{4n_{local}}$ of the local communication radius, r, on an amorphous computer. Whether such a resolution is achievable is a different question. For n_{local}=15. this implies a resolution limit of $.05r$, which is far below that achieved by the gradients.

6.2.2 Analysis of Axiom Robustness

The previous section showed that gradients produce reasonable estimates of distance when the expected local neighbor n_{local} is greater than 15. However the distance estimates are not error-free, and the average error is on the order of $0.4r$ for $n_{local} = 15$, where r is the local communication radius. The axioms are designed to compensate for this by using local comparisons of gradient values and creating new gradients to minimize error specifically in the area where the crease is expected. The goal of an axiom is to produce a straight crease of width $2r$ at the appropriate location. The

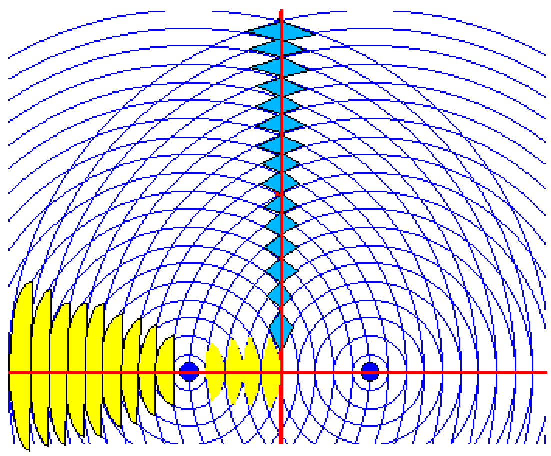


Figure 6-5: Interference between gradients from two sources. The concentric bands represent the radially-symmetric uncertainty in distance estimates from a gradient from a single source. The composition of two gradients causes the error to vary spatially.

shapes and patterns from chapter 5 suggest that the axioms manage quite well. In this section I provide an analysis of how well the axioms achieve this goal, assuming $n_{local} = 15$.

Axioms 2 and 3

I will start by analyzing the behavior of axiom 2. Axiom 2 produces a crease that is the perpendicular bisector of the line between two points A and B (for a review of axiom 2 see chapter 4). At the cell level this is achieved by the cells in A and B creating two gradients, and then all cells compare their values of these gradients to see if the difference in value is less than 2. This threshold is determined by the desire to create a crease of width $2r$.

Thus axiom 2 depends on the composition of two gradients. Although a single gradient produces estimates whose error is not correlated with orientation about the source or distance from the source, when two gradients are composed this is no longer true. Instead one gets a kind of spatial "interference pattern" between the two gradients, as illustrated in figure 6-5. The concentric bands around each source represents the uncertainty of the distance estimate; the width of the band is the expected error in the distance estimate. Along the line between the two sources the concentric rings look almost parallel and intersect to create large overlap regions. Along the bisector, the overlap regions are not so bad. In both cases the overlap regions increase in size as one moves further away from the sources. The overlap region signifies the region within which a cell may exist — the larger the region, the larger the uncertainty in the position of the cell.

We can see that the error in position of a cell depends not only on the error in

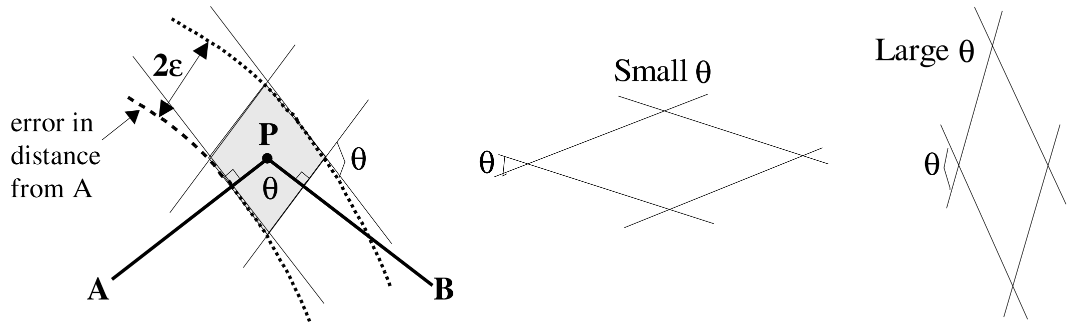


Figure 6-6: Area of uncertainty as a result of interference between gradients with error.

the gradients but also in the position of the cell relative to the two sources. This uncertainty can be characterized as follows.

Theorem 2: *The expected error in the position of a cell P relative to two point sources A and B is determined by the area of the parallelogram with perpendiculars of length* 2ϵ *and internal angle* θ*, where* ϵ *is the expected error in the distance estimates from A and B and* θ *is the angle* $\angle$ *APB. The area of the parallelogram is* $\frac{(2\epsilon)^2}{\sin\theta}$.

Proof: Let the distance estimate from A to P be a and the distance estimate from B to P be b. The position of P is determined by $\triangle$ ABP. However because of uncertainty in distance estimate from A, P lies within the circular band $a \pm \epsilon$. Similarly P lies within the circular band $b \pm \epsilon$. Thus P can be anywhere within the intersecting region of those two bands (figure 6-6). This overlap area can be approximated as a parallelogram, when the distance of P from the sources is large compared to ϵ and the curvature of the bands can be ignored. The perpendicular distance between the opposite sides of the parallelogram is 2ϵ, which is just the width of the band. The internal angles of the parallelogram are determined by the angle of incidence between the two bands.

$$\text{Area of paralleogram} = base \times 2\epsilon \tag{6.11}$$

$$\sin\theta = \frac{2\epsilon}{base} \tag{6.12}$$

$$\text{Therefore area} = \frac{4\epsilon^2}{\sin\theta} \quad \text{q.e.d.} \tag{6.13}$$

The area of the parallelogram is minimized when θ is 90 degrees (square). When θ is very large or very small, the bands appear to be parallel to each other, which explains the large overlap regions along the line AB in figure 6-5. Along this line curvature plays a role in limiting the overlap near A and B, but as one moves further away the overlaps increase rapidly.

Along the bisector, the behavior is quite good by comparison (figure 6-5). Near the line AB, θ is large but most of the error is not along the width of the crease. Only as one moves further away from A and B does the error in the width of the crease start increasing (when θ is small). The distance at which a particular angle occurs is proportional to the distance AB. Hence we expect the accuracy of axiom 2 to decrease as the ratio of the crease length to distance between sources is increased.

A similar analysis can be made for axiom 3. In axiom 3 the crease formed is the bisector of the angle between two input lines. At the cell level the crease is generated using the same local rules as axiom 2; the input lines generate gradients and cells test if the difference in gradient values is within the threshold. In this case the bands of uncertainty in distance estimates are parallel to the input line and hence the overlap regions are exactly parallelograms. For axiom 3 θ is determined by the angle between the two input lines.

Theorem 3: *The expected error in the position of a cell P relative to two line sources L1 and L2 is determined by the area of the parallelogram with perpendiculars of length 2ϵ and internal angle θ, where ϵ is the expected error in the distance estimates from L1 and L2 and θ is the angle between L1 and L2*

When the angle is small the error is small along the width of the crease. As the angle between the line increases, the uncertainty along the width of the crease increases. This starts to become a problem as the angle between the lines becomes obtuse. If the angle between the lines is 180 degrees then the axiom may not work at all — the bands are close to parallel and the majority of cells think they belong in the crease. A rule of thumb would be to try to avoid large obtuse angles.

Experiment: The above analysis suggests that the accuracy of the crease produced by axiom 2 will vary with the ratio of the distance between sources to the length of the crease. Here I present simulation experiments that show the accuracy of the creases produced for different distances between the sources ($n_{local} = 15$). In each experiment two sources s1 and s2 are chosen along the diagonal c1 c3 (i.e. line $y = -x + 1$) such that the sources are symmetric about the center of the crease. Using axiom 2 with these sources should produce a crease along the c2 c4 diagonal (i.e. line $y = x$). By varying the distance between the two sources (and keeping the length of the desired crease constant) we can see how the distance between sources affects the crease.

In order to measure the accuracy of a crease, I define an *ideal crease*. An ideal crease is such that it occupies a width of exactly $2r$ around the desired line. In terms of cells, we can describe this using three criteria:

1. The best-fit-line of the cell positions is the desired line.

2. Within distance $2r$ of the best-fit-line, all cells belong to the crease.

3. Outside distance $2r$, no cells belong to the crease.

Criterion 1 tests the position of the crease while 2 and 3 test the straightness and uniformity of the width of the crease. For example a crooked or wide crease would do badly in criteria 2 and 3. However we do not expect the width of the crease to be exact and a small amount of non-uniformity along the length of the crease is acceptable, as long as it can be bounded by some error margin. Based on the expected error in the gradients for $n_{local} = 15$, the crease width is expected to vary between $1r$ and $3r$.

In order to test the width of the crease I plot the percentage of cells within $1r$ that are not part of the crease, and I plot the percentage of crease cells that lie outside $3r$. For an ideal crease the first value would be 100% and the second would be 0%. I also separately plot the average and maximum deviation of the crease cells from the best-fit-line. Both plots give us an idea of the width of the crease. In order to determine the accuracy of the position of the crease, I plot the difference in slope between the best-fit-line and the desired line.

The graph in figure 6-7(a) shows that if the length of the desired crease is less than twice the distance between sources, then axiom 2 creates a crease of width close to the ideal. The width of the crease spreads as the distance between the sources gets smaller, relative to the crease width; a larger percent of crease cells lie outside $3r$. This is exactly as predicted by Theorem 2. A rule of thumb would be: given two points A and B, construct creases that extend no more than the distance between A and B above and below the line AB. The graph in figure 6-8(a) shows that the best-fit-line of the cells is extremely close to the desired line, even when the sources are close together. These experiments show that axiom 2 predicts the desired line with very high accuracy.

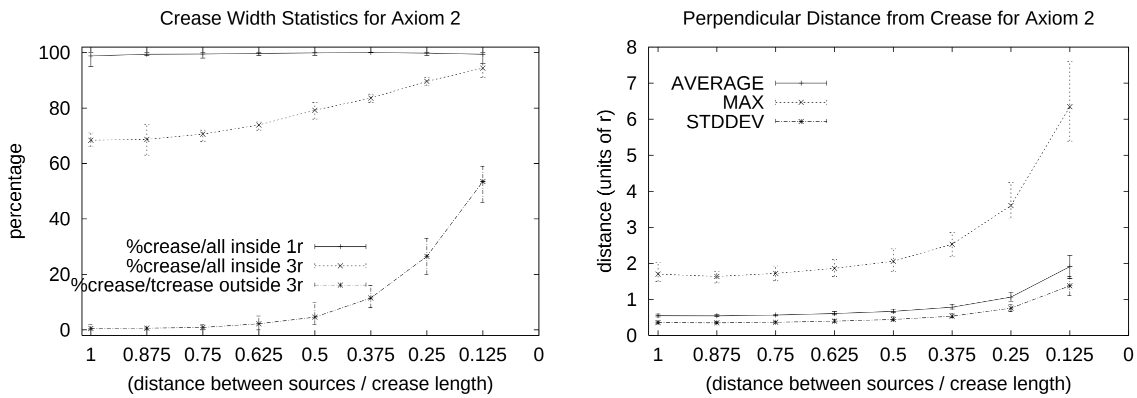
Axiom 1

In axiom 1 the crease formed is the line between two points A and B. Figure 6-5 and Theorem 2 suggest why it would be a bad idea to use two gradients to find the line between A and B — the error is the worst along the line AB and gets worse as the distance between A and B increases. Instead we use a single gradient and therefore the performance of axiom 1 is not affected by the compositional error. For a single gradient the error is radially symmetric and not strongly correlated with distance. The cells in B create a gradient and and the cells in A compete to grow a crease towards B. Each cell along the crease compares local neighboring values of the gradient and chosen the neighbor with the lowest value to continue the crease.

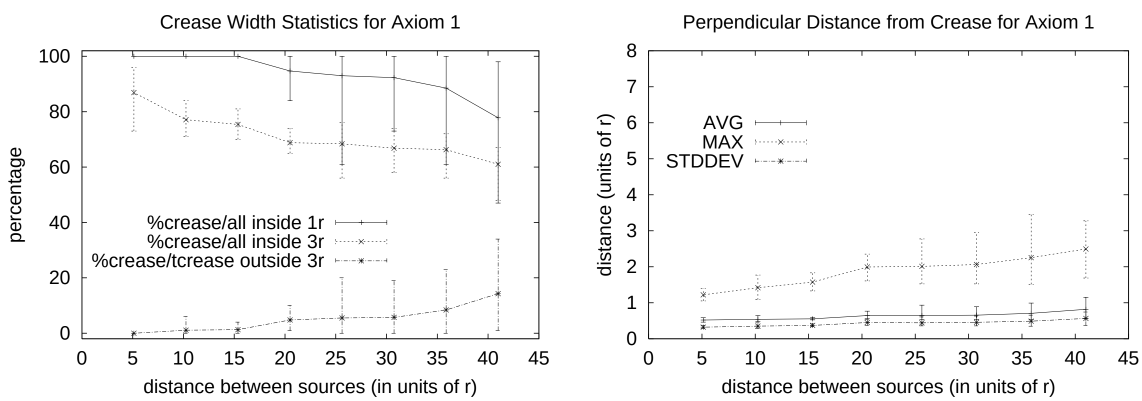
Daniel Coore has provided an extensive analysis of the affect of random distribution of cells on the crease formation using similar methods in his thesis [13]. He has shown that the crease is most affected by local variations in the density. Therefore if the variance of the density is reduced (for example by limiting the overlap of cells), the geometric properties of the line improve. The higher the variance, the more the crease tends to wiggle. Axiom 4 has essentially the same behavior, but follows a gradient with parallel bands.

An interesting feature of axioms 1 and 4 is that unlike axioms 2 and 3 they produce creases that guarantees connectivity between inputs A and B. The crease grows around holes, sacrificing straightness for connectivity. Coore has shown that line formation using GPL is extremely robust and rarely does it fail to form a connected crease.

(a) Axiom 2 Experiments



(b) Axiom 1 Experiments



Legend:
%crease/all inside x = percent of cells within width of x that are part of the crease,
%crease/tcrease outside x = precent of crease cells that are outside width x.

Figure 6-7: Experimental results on the crease widths produced by axioms 1 and 2, as a function of distance between the sources.

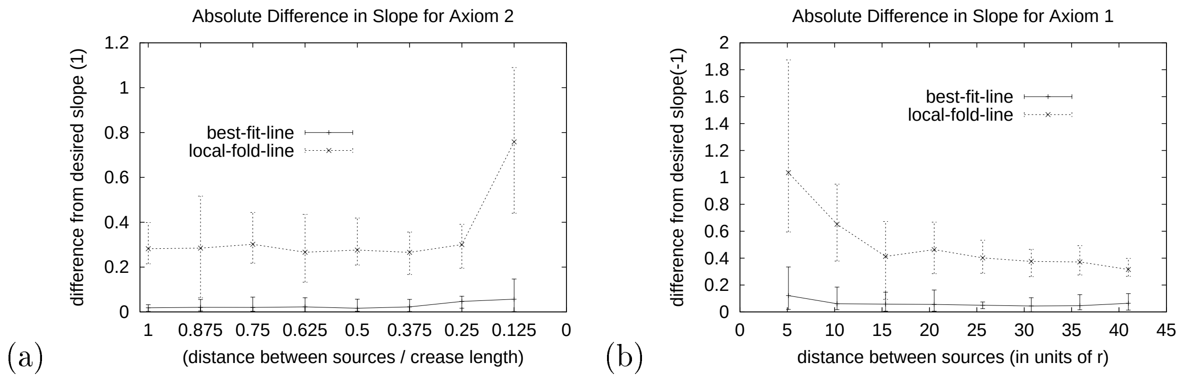


Figure 6-8: Experimental results on the accuracy of the creases produced by axioms 1 and 2, as a function of distance between the sources. Graphs compare the slope of the desired line to the best-fit-line and the locally determined fold direction.

Experiment: Similar experiments are performed on axiom 1, as were done on axiom 2. Two sources are chosen along the c1 c3 diagonal, such that the sources are symmetric about the center of the sheet and a crease is grown from source 1 to source 2 using axiom 1. The length of the crease is equal to the distance between the sources. As the distance between the sources increases, there is more chance for the crease to wiggle around. However the width of the crease is not expected to change much.

The graphs in figure 6-7(b) show that as the length of the crease increases, there is more variability in the crease. The region within $1r$ is not full of crease cells because the crease wiggles around more. And crease cells are found beyond $3r$ with more frequency. Therefore axiom 1 is likely to be more robust when the crease length is small. Still as the graph in figure 6-8(b) shows the best-fit-line does predict the desired crease reasonably well. An important point to note is that in none of the experiments (over 100) did the axiom fail to form a connected crease.

Intersect and Region Formation

In each case the axioms attempt to create gradients with a crease of width 2r. The intersection of two creases produces a point of area on average $4r^2$. By choosing a reasonable local density, the probability of having no cells in this area becomes extremely small. Therefore the probability of intersect failing due to the absence of cells is very low. Region formation is also extremely robust. Axiom 1 and 4 generate creases that are guaranteed to be connected, therefore a leak is unlikely. Axioms 2 and 3 do not guarantee connectivity; however the main reason why a crease may be disconnected is because of the absence of cells at that location. In that case the region can not leak through the hole because there are no cells to transmit the gradient.

6.2.3 Analysis of Fold Robustness

As previous section showed the axioms produce good creases — the position of the crease is reasonable accurate and the creases are reasonably straight. For folding a crease, the cells must also be able to correctly predict the orientation of the crease. As mentioned in section 4.2.4, this is done in two steps. Each cell first determines the fraction of its neighbors that are part of the crease, we call this value the strength of the crease. The strength is highest along the center of the crease and less towards the edges. Each cell then collects the strengths of its neighbors and chooses the highest and lowest neighbors to estimate the orientation of the crease. The average of the locally determined orientations is then compared to the actual direction of the crease.

In figure 6-8 we compare the locally determined crease line with the best-fit-line and the ideal crease line for the previous experiments on axiom 1 and 2 creases. The graphs plot the average absolute differences in slope between the ideal crease, the best-fit-line and the locally determined line. The graph shows that the differences in slope are small; the cells can reasonable determine the fold orientation. However the locally determined fold line is not as accurate as the best-fit-line. There may be other mechanisms that can be used at the cell level to predict the orientation of the fold.

6.3 Random Cell Death

One of the advantages of designing for random distribution of cells, is that even if a small fraction of cells randomly die, the result is still a random distribution. Therefore the algorithms designed for random distributions have some amount of inherent tolerance to random cell death. If at the beginning of an experiment we randomly kill some percentage of cells, then that is equivalent to decreasing the density of the cell distribution. Therefore the performance will be affected if the expected local neighborhood goes significantly below 15.

All of the local primitives are designed with a certain amount of redundancy; the overall behavior is the average of many cells and never dependent on an individual cell. This is done mainly for achieving low error on randomly placed cells, but also provides redundancy against random cell death. Gradients are extremely robust to random cell death. The source cells start the process however they are no longer needed once the initial message is sent. The remaining cells just propagate information and any individual cell is not critical to the success of the gradient. Axioms 2 and 3 have cells autonomously decide if they are part of a crease and their death has little affect on the overall process. Only in axioms 1 and 4 is there some dependence on a single cell (the growing point) when the crease is growing. However the dependence on any one cell is for a very brief period of time and the probability that the cell die at exactly that moment is extremely unlikely. Gradients and axioms are only vulnerable when the an entire source or input dies. In general the activity constantly shifts around within the sheet and much of the information is transient. There is no fixed hierarchy or centralized control which makes it much more difficult to disrupt the shape formation.

Points and lines are the only sections that have long term roles and the sheet formation is vulnerable to the death of entire points or lines. Therefore it is important to have sufficient cell density to ensure that points and cells have reasonable numbers of cells with high probability. Within a point or line all cells are equal and the behavior of a point is the average of all the cells within the point.

Chapter 7

Scale-independence, and Other Analogies to Biology

An interesting global property of the Origami Shape Language is that it is "scale-independent". Scale-independence implies that a program encodes only the shape, not the size. Thus one can create the same shape at many different scales without modifying the program. In OSL, as in origami, the final size of the global shape is determined by the initial size of the sheet. This provides an insight into how global shape and proportion can be controlled at the cell level. Using this insight I show how a single OSL program can generate many related shapes, in the manner of D'Arcy Thompson's famous coordinate transformations. The final shape is determined by the initial shape of the sheet.

The language provides many other interesting insights into the relationship between local and global descriptions of behavior. Many of these properties have analogies to biological systems. Whether these analogies are coincidental or indicative of something fundamental will require a great deal more research. But the analogies do dispell some of the mystery behind how complex morphology can arise from modest beginnings and provide theories that can be a basis for biological experimentation.

7.1 Scale-independence

Scale-independence in biology (also known as size-invariance) refers to appearance of the same structure at a variety of scales. Extremely complex structures such as lungs, kidneys and the digestive system appear in a wide range of sizes.[1] Embryos of the sea urchin develop normally over ten-fold size variations, and in the case of hydra a fragment one hundredth the volume can give rise to complete animal [71]. Species of the *Drosophila* vary over five-fold in size, but the shape and proportion of the body parts is highly conserved, as is their DNA [42]. Many of these cases would suggest (especially in the case of the embryos) that the processes for forming morphology

[1] I still remember being amazed when I dissected a frog in high school that the organs were so perfect yet so small and so similar to the life-size human anatomy diagram on the wall.

are capable of scaling to different sizes, without modification of the DNA. However biologists currently have very little understanding of how this is achieved at the cell level. Wolpert proposed the idea of positional information and "balancing" gradients to explain scale-independence in the initial patterning of the hydra and sea urchin [71]. He used a small number of gradients to create a coordinate system (which he called positional information) and used different comparisons (e.g. $a = 2b, a = b/c$ where a, b, c are gradient values) to generate patterns. One problem with this model is that generating complex scalable patterns requires very complex comparison functions and significant precision in the original coordinate system.

The Origami Shape Language is scale-independent — the same global program (and hence the same cell program) can produce a shape at different scales. Figure 7-1 shows cups formed from 2000, 4000 and 8000 cells (with the same cell density per unit area), all using the program in figure 3-4. Of course the scalability is not unlimited. But one can conceive of creating a larger cup simply by starting with a larger sheet. Most artificial pattern formation techniques are not scale-independent; they either encode a fixed unit length that determines the pattern size or the pattern itself changes as the number of cells is increased [13, 58, 28]. In OSL, only the crease width is determined by the local communication radius; the global shape scales with the number of cells.

At the cell level, scale-independence is achieved by always using relative comparisons of gradients, never absolute thresholds, which is similar to Wolpert's model. However instead of many complex comparators, we use three simple comparisons: 1) testing if two gradients are equal 2) locally testing the direction (tropism) of single gradient using a less-then comparator and 3) testing for the existence of a bounded gradient for marking regions. At the global level scale-independence is a side-effect of choosing origami as the global description language. Origami encodes the formation of a shape without reference to the size of the sheet. The sheet starts with only a boundary. Each operation (fold) is relative to the current boundary and creates new "boundaries". Thus complexity is generated by recursively applying simple operations, while still remaining relative to the original boundary. At the cell level, this translates to using many gradients, simple comparators and local state as opposed to a single coordinate system.[2]

Thus we are able to create highly complex structures without reference to size, demonstrating both the power of Wolpert's idea as well as the power of origami.

7.1.1 Other Types of Scale-independence

Both my model and Wolpert's model assume that gradients extend across the whole sheet, and this is important for scale-independence to work. In my model gradients can be restricted within a smaller region for forming substructures, therefore not all gradients need to travel across the sheet. In a biological setting one would expect the

[2] Current research on the *Drosophila melanogaster* embryo suggests that the initial body segmentation pattern is created by many gradients, and that a single gradient is used to determine no more than four thresholds [55].

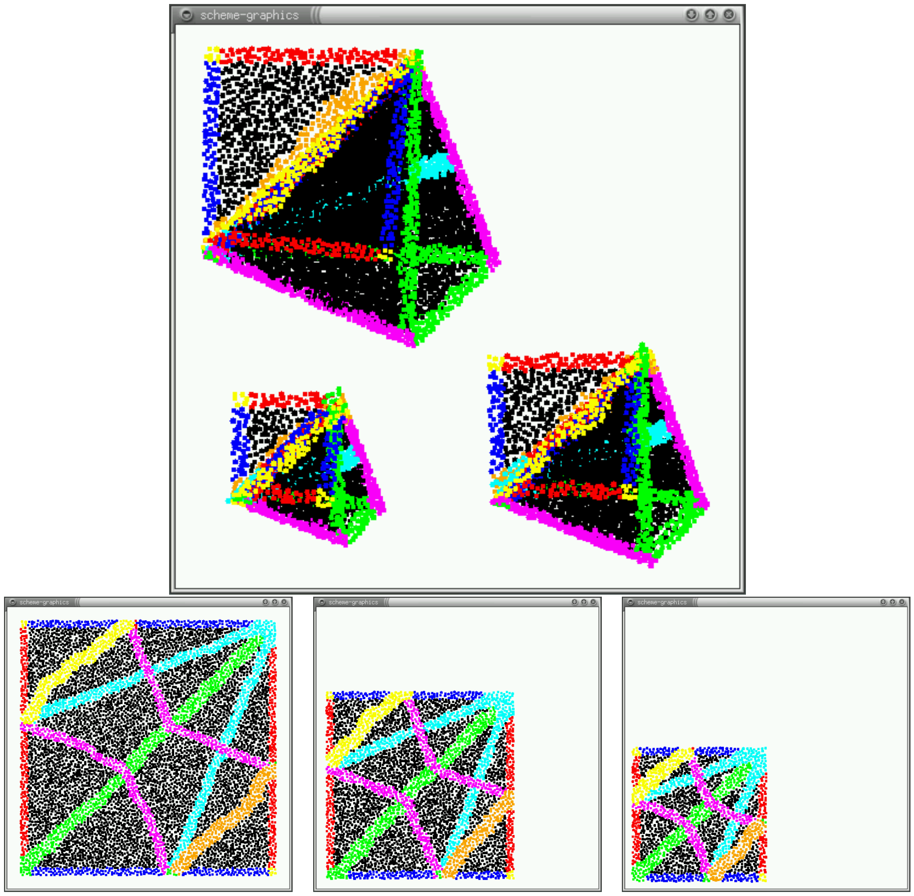


Figure 7-1: OSL cup generated on different sized sheets (8000, 4000, 2000 cells at constant density). In each case the cell program is the same but the size of the cup scales with the size of the sheet.

extent of a gradient to be related to the amount of morphogen being created and this would place an upper limit on the scalability of both models.[3]

There are likely to be many other mechanism by which scalable patterns can be achieved. For example it is not clear whether relative comparisons are necessary for creating a scale-independent pattern. One possibility is that as the size of the initial substrate increases, the number of source cells may increase. In this case, one could create a scalable pattern from an absolute threshold — the increase in morphogen emitted would move the absolute threshold further away from the source thus giving the appearance of a scaled response. This would violate Wolpert's claim that bidirectionality is a necessary condition for scale-independence [71].

Another possibility would be that the pattern formation is not scalable at all.

[3]It is possible that real systems compensate for this — for example, it seems that a great deal of patterning occurs when an embryo is small, which would allow morphogens to travel a significant distance along the embryo.

Rather the pattern is formed on a substrate of fixed size and the size of the final structure is determined by the amount of growth. For example, although *Drosophila* species may vary considerably in size, the embryos during the pattern formation stage may not — which would imply that the pattern formation rules need not be scalable. This would however not explain how certain embryos, such as the sea urchin, develop normally over many different starting sizes.

The above ideas suggest many possible biological experiments. In the *Drosophila melanogaster*, Nusslien-Volhard and Weischaus showed that the head, thorax and abdomen segmentation boundaries are affected by thresholds of the *bicoid* protein generated from source cells (maternally determined) at the anterior pole. However *Drosophila* species occur over wide range of sizes. It is likely that the initial patterning proteins and genes are conserved across many of these species. If one could compare two species of different sizes, one could ask several questions: a) are the embryos same size when initial pattern occurs? b) if not, do the segments scale proportionally with the embryo size? c) how does the extent and concentrations of the *bicoid* gradient compare in the different species and do segments occur at relative or absolute concentrations of *bicoid*? Answers to each of these questions would shed light on how the initial pattern formation happens and how it relates to theories of scale-independence.

7.2 Shape Transformations a la D'Arcy Thompson

One of the insights derived from origami is that if the shape is programmed as a construction relative to the boundary, then size can be determined by scaling the boundary. This idea is very powerful because by *asymmetrically* changing the boundary, one can create many related shapes from a single program.

Figure 7-2 shows the inverter program from section 5.2 run on initial sheets of different shapes. If the sheet is stretched along the y-axis, the inverter pattern stretches accordingly. If the sheet is asymmetric, then the pattern also follows the contours of the border. Because the inverter is formed by dividing the sheet into regions, one can think of changing the boundary as transforming the coordinate system. However this idea is not limited to patterns; a cup can be created from a rhombus shaped paper resulting in a short wide cup or a tall thin one.

The idea of shapes in biology being related by coordinate transforms was one of the many fascinating theories presented by D'Arcy Thompson in his book "On Growth and Form" [64]. He observed how the forms of many related animals (crabs, fish, skulls) could be plotted on Cartesian coordinates and transformed into one another by stretching along different axes (see figure 7-3). He claimed that these relationships extended not only to superficial characteristics, but also internal organs. Although he did not investigate the mechanisms for the transformations, he believed "that a comprehensive 'law of growth' has pervaded the whole structure in its integrity and that some more or less simple and recognizable system of forces has been in control". Different growth rates of cells along different axes could produce such patterns. The Origami Shape Language suggests another mechanism — changing the shape of the

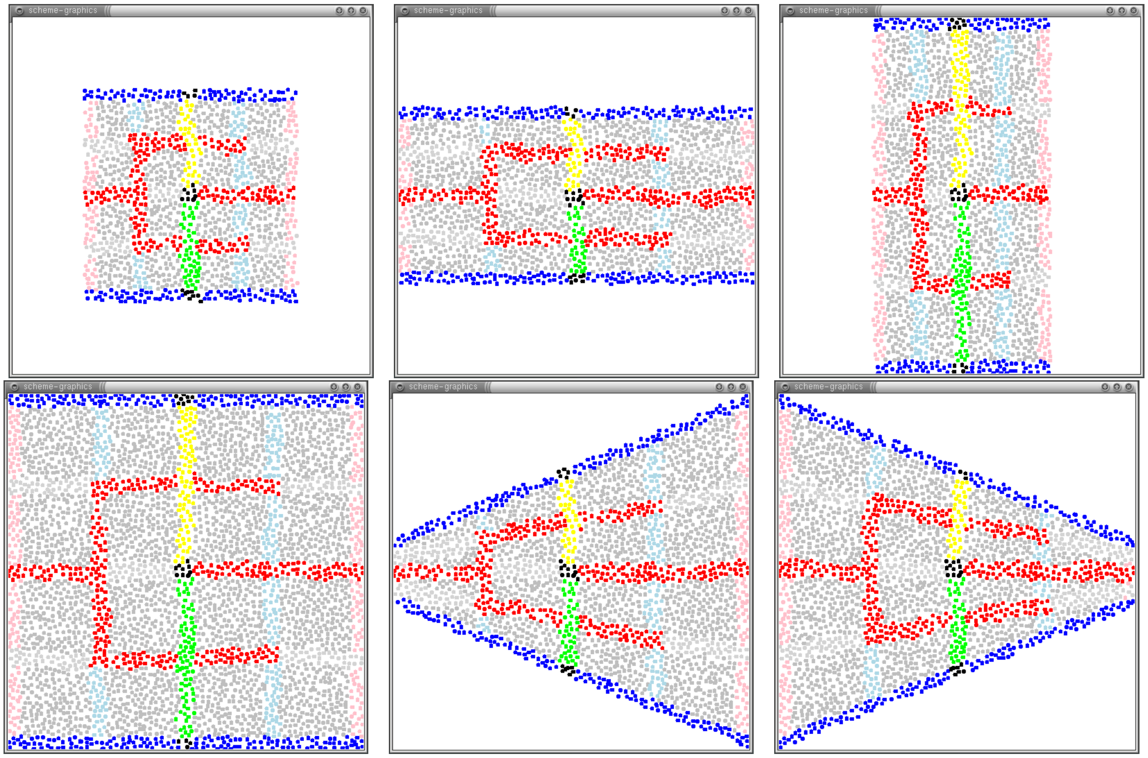


Figure 7-2: The inverter program executed on differently shaped sheets.

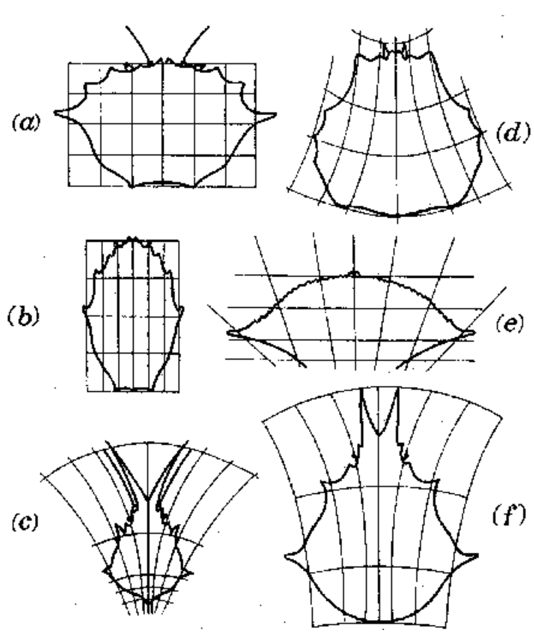


Figure 7-3: The carapaces of related crab species on a common coordinate system. From *On Growth and Form* by D'Arcy Thompson [64].

initial boundary.

The inverter transformations in figure 7-2 are very similar to Thompson's crab transformations in figure 7-3. Using OSL one can generate many interesting related shapes, including shapes with complex internal patterns or proportions that scale differently in different dimensions. These figures are caricatures and are not meant to model biology. Rather the object is to show that related forms can be achieved through local patterning mechanisms.

As Thompson pointed out, related species provide a great example of related forms whose construction we might expect to be related.[4] Figure 7-4 shows the heads of several related species of *Drosophila* [42]. The proportions scale in interesting ways; not all regions of the heads scale in the same way. This poses some questions, for example is the center region fixed in size or does it scale with some other dimension of the head? And what are the observable differences in the formation processes of the head? Figure 7-5 is an OSL caricature of the heads. In each case the pattern is generated by the same cell program, the only difference is the shape of the initial sheet. In the OSL program the width of the center region is proportional (half) to the width of the sheet and achieves similar scaling. Although the caricature is simple, one can see that any pattern formed within any of the major regions (i.e. relative to its boundaries) would scale automatically as one scaled the overall sheet. Experiments on two closely related Hawaiian *Drosophila* species, with very different looking heads, has shown that there are only a small number of genes, probably 10, that are responsible for the difference in head shape [65]. The OSL program demonstrates how significantly different looking shapes could be produced by the same cell program and suggests that one possible function of these genes may be to determine the growth along the major axes of the head.

The power of D'Arcy Thompson's idea is that it shows a global way of approaching related shapes, without getting muddled by the internal complexity of the shape. In his words "it will also demonstrate the fact that a correlation which had seemed too complex for analysis or comprehension is, in many cases, capable of very simple graphical expression" [64]. I believe that the Origami Shape Language shows that related shapes can also be approachable at the local level. Visually dramatic global differences do not necessarily translate to large differences in a cell program. By experimenting with many different self-assembling mechanisms with different global properties ([13, 4], one can form theories that help recognize key differences in the developmental sequence of related shapes, rather than get intimidated by the overall complexity of the formation.[5]

[4]Note that this is not necessary. Even in OSL one can write more than one OSL program that creates the same shape. Therefore it is important to not only look at the genes and end structure, but also at the developmental sequence.

[5]Independently, origamists too have been studying related shapes. Kawasaki has shown that the *Orizuru*, the traditional origami crane, can be created from rhombus and kite shaped paper, resulting in related orizuru with differently proportioned wings, tails and heads. By understanding the essence of the folding sequence one can even create orizuru from rectangular paper (asymmetrical wings) or recursive orizuru (orizuru embedded in the wing of a larger orizuru) [38].

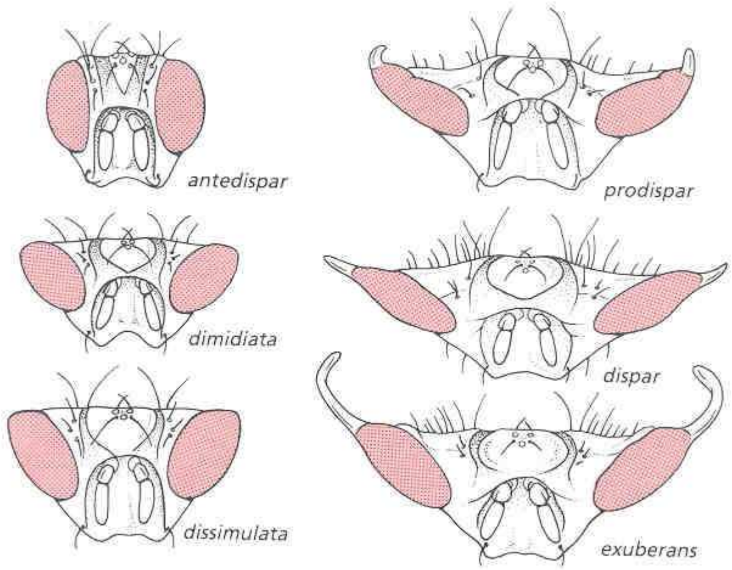


Figure 7-4: The heads of related species of *Drosophila*. From *The Making of the Fly* [42].

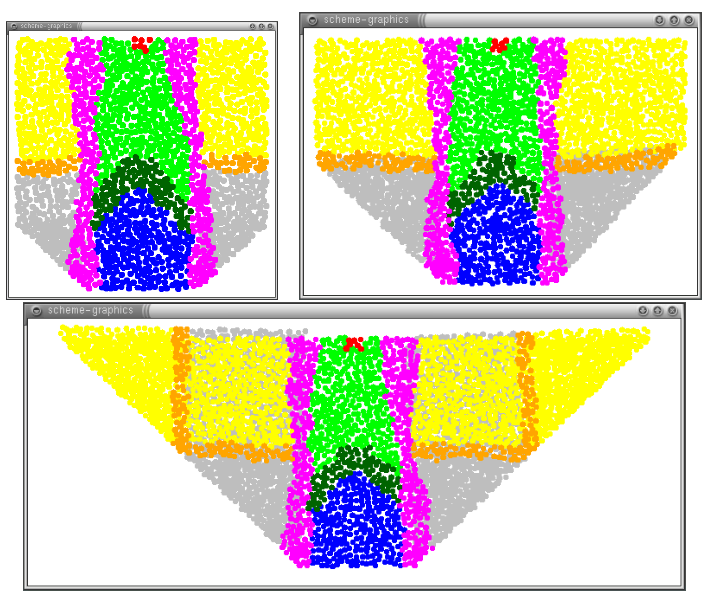


Figure 7-5: A caricature of how the *Drosophila* head proportions vary. The width of the center region is proportional to the height of the sheet, therefore it does not extend as the length of the sheet is increased.

7.3 Other Properties

The language exhibits several properties that illuminate the relationship between local and global descriptions of behavior. Many of these properties resemble biological systems. This artificial system gives us a means for experimenting and exploring the relationship between global complexity and local processes to see what can be achieved and how it is achieved. This provides a framework for asking questions about (and recognizing interesting features in) biological systems.

Notice that the computational model of the cells is already influenced by what we think biological cells are capable of. For example, all cells have the identical program (like having identical DNA) and differ only in small amount of dynamic state. Initially the sheet is mostly homogeneous with only a few initial conditions: border and a polarity. There are no global coordinates or unique identifiers per cell, and the fates of cells are determined through the process of differentiation. These similarities were chosen to facilitate analogies with biological systems as well as motivated by manufacturing considerations. In this section I discuss similarities that have arisen as a result of the choice of origami as the global representation of shape.

7.3.1 Local Code Conservation

If we look at the cell program generated by compiling the global shape construction, we see that the majority of the code is dedicated to implementing primitive functions of the cell (local communication, gradients, collecting and comparing neighbor values) and implementing the local rule logic that captures how simple patterns are constructed. Only a small piece of code specifies the sequence, arguments and local state that determine the final shape. Hence the majority of the cell program is conserved across all shapes. Two very different shapes would appear to have practically the same code, even if the process of formation looked very different. This is suggestive in light of the fact that DNA is highly conserved across all living things. The fact that humans and chimpanzees share 99% of their DNA (or that humans and yeast share 60%) is mysterious only because we do not know how to relate the difference in code to the differences in morphology. The fact that it is possible should not be mysterious.

7.3.2 Gradient Reuse

In the compilation process it is possible to reuse gradient names many times throughout the program. This is because the use of a gradient is short-lived. A cell can simply flush the value of the gradient after some period of time. The total amount of permanent state in a cell is in fact quite small (mostly booleans), and corresponds to the distinct features in the pattern or shape.

In a biological system this would be analogous to reusing the same proteins or morphogens over and over again at different times during the development sequence. This suggests the possibility that there may not be very many morphogens in total. This also complicates the problem of probing an embryo since any one gene and any one protein may have many possible unrelated functions during the course of development; knocking out a gene may only reveal the first and not necessarily most important function of that gene.

7.3.3 Repeated Structures

As a hypothetical example, say that we want to create an origami hand with five fingers. Then we would first code a folding sequence that divides the sheet into a palm with four flaps (one for each finger), and then define a single procedure for patterning a finger and call it within each flap. Each flap could be of a different length and width, and the finger pattern/proportions would automatically scale to the finger shape. This example illustrates several interesting points. First in OSL one can compactly capture repeated structures, or rather repeated "formation sequences". The end structures may look slightly different, but that difference can be captured in the initial conditions or "context" within which the procedure is called. This is the same idea as creating related and scaled shapes, but applied to substructures. We can also consider how mutations in the OSL code would produce extra fingers. For example a mutation in the high level code that created six flaps, instead of five, would be sufficient to create an extra finger without any modification to the finger pattern procedure.

The point of the example above is not to explain how a hand is formed, but rather to illustrate that it is possible to capture the "relatedness" of form separately from the "differences" and to suggest that this has advantages such as the compactness of code and the ability to add new substructures through small mutations. Mutations such as the bithorax-postbithorax mutation in the *Drosophila* suggest that this is not such a far-fetched idea. Two mutations in the regulatory region of a single gene, the ultrabithorax gene, convert a balancing organ (haltere) into a wing, creating a four-winged fruit fly. This suggests that there is some "procedure" (set of genes) that creates a wing and some initial conditions that are met by the pre-haltere region, such that it is possible to "invoke" the wing procedure instead of the haltere procedure. It shows that the gene controls both the left and right choices of wing or haltere.

By testing for defects that affect one substructure but not another, or defects that affect all related structures, one can begin to understand which parts of the pattern are under independent control and which parts are common. For example, one could investigate if the relative lengths of the different legs of some insect are related or independently controlled, or if the segment proportions of all fingers in the hand are under common control.

The abstract idea of capturing common formation sequences or common "algorithms" in a procedure is very attractive, but there are some questions as to how a procedure could be represented in DNA. For example, what does it mean to pass in arguments to a procedure? And what does it mean to have local variables (local names) within a procedure? Within-region provides us one example of a solution — the arguments can be implicit (the shape of the region is an implicit argument) or be global names with different values in the different regions. The gradients and effects of the local rules are bounded by the region, thus preventing interference between procedures invoked in disjoint regions. In the case of the *Drosophila* mutation example, there are compartments defined by selector genes and the wing and halteres form from two distinct compartments - the mutation affects the choice of the cells in that compartment to form a wing or haltere. However one can think of other structures, such as branching structures, where a procedure that created a branch would not need a compartment. Instead if gradients are limited in extent the interference could be eliminated spatially[13].

7.3.4 Topology versus Geometry

In the Origami Shape Language there are essentially two means of creating line pattern: axiom 1 which generates a crease by growing a crease from one point to another, and axiom 2 which generates a crease through the comparison of multiple gradients. These two axioms represent fundamentally different strategies of generating structure — one guarantees connectivity while the other favors geometry. In axiom 1 information is passed from cell to cell which guarantees that the line remains connected. If there is a large hole of cells, this crease will tend to grow around it. Thus its favors a connected crease over a straight one. On the other hand, in axiom 2 cells autonomously decide whether or not they are part of the crease. Even when cells are randomly distributed the crease formed is very straight. However if there are cells missing in areas that would have been part of the crease, then the final crease is discontinuous but still straight. During development, at different times topology or geometry may be important and the relative important is partly revealed by the strategy chosen. For instance during gastrulation the gut grows from one end to the other guaranteeing connectivity between the mouth and anus. Similarly the formation of neurons may involve growing towards a source. However during the segmentation of the body into regions, cells may autonomously decide which region they belong since connectivity is not an issue.

7.3.5 Symmetric Structures through Folding

Using OSL one can create accurate symmetric patterns by folding and then unfolding (for example the tessellation patterns from chapter 5). An interesting and speculative question is whether there are examples in nature where the formation of symmetric structures or patterns takes advantage of folding. For example, butterfly wings are formed in a folded state. Does this have an effect on the final repeated patterns formed on the wing? A very different example is the formation of insect limbs, that also happens in a partially folded state. Does this help ensure that the various segments of the legs match in size? It is certainly known that bone and muscle often develop in unison so that the muscle is not only the correct length but also connected to both ends. The idea that folding may play a role in development of structure is still very speculative and following up the examples mentioned is left to future work.

Chapter 8

Conclusions and Future Work

8.1 Discussion

In this thesis I have presented a new approach to designing local behavior for specific global goals, by using two separate programming paradigms at the global and local levels. At the global level the approach is that of folding a continuous sheet and takes advantage of knowledge in geometry. At the local level the paradigm is inspired by developmental biology, where there are many examples of robust and complex shape formation accomplished by identically-programmed unreliable cells.

Gradients have been around within biology for a long time and similar primitives (pheromones, for example) have been used to coordinate collective robots, and more recently form branching structures on a reconfigurable robot [47, 27]. I expect that the analysis in this thesis will prove useful in many domains. However the approach to designing local behavior is still dominated by either trial and error design or the use of evolutionary methods to generate complexity. This is partly because many of the local rules, such as ant-based rules, are difficult to characterize and even simple compositions can cause un-intuitive behavior. Another problem is that without a global paradigm it is difficult to determine how to compose the local rules to produce predictable global behavior.

In this case we take advantage of knowledge in a different discipline of how to construct global shape. The Origami Shape Language specifies how shape can be constructed from a continuous sheet, by applying a set of axioms. Then each axiom is translated to cell level behavior. This is a very different approach than trying to directly generate cell level behavior from a picture of the end shape, which throws away a lot of knowledge.

8.2 Future Work

8.2.1 Other Constructive Languages

There are many other constructive languages and one question is whether constructive global descriptions are inherently easier to compile to local behavior. By constructive

I mean that the goal is described as a process instead of an end state.

One example of a constructive language is *digital circuits*. Say that we wanted to design a substrate that self-assembles to perform different computations. We could describe the computation as a digital circuit, constructed from a small set of gates (AND OR NOT) with simple rules for connecting the gates. Then we could then try to find local mechanisms for self-assembling each type of gate and implementing the connection rules. This is very similar to the approach used in this thesis. This allows us to take advantage of: a) Our knowledge of how to decompose an arbitrary computation into a set of simpler units and b) Our database of digital circuit designs. If this is possible, programming the substrate to become an adder may be as simple as looking it up a good adder in a circuits book.

8.2.2 Biological Experiments

In chapter 7, I discuss many interesting properties of OSL that have analogies to biology, such as scale-independence, related shape and the reuse of gradients. This system shows that it is possible to generate many different types of shapes and the same shape at different sizes without any modification to the cell program. It also suggest how — by recursively subdividing the space relative to the boundary. If the cell programs do not differ, then methods such as genetic analysis are not likely to reveal much information. Instead observations of the process of development and comparisons between processes in closely related species may give us better insights into how the morphology is created. The language does not model biology, but it does serve as an example of how certain properties can be achieved by cells and provides many theories that will be interesting to pursue.

8.2.3 New Metaphors

So far the work on self-assembly has explored a very limited space. Biology suggests many other metaphors for creating complex structures.

1. **Growth:** Growth is key piece to generating structure in embryos, and even there gradients may play a role in directing the growth [42].

2. **Deposition:** A different process is structures created through the deposition of materials such as collagen. Another example is the construction of termite mounds.

3. **Cell Death:** Cell death plays an important role in the formation of structures such as the *Drosophila* eye, but also in the formation of 3D structures by first creating scaffolds and then using programmed cell death to remove material between scaffolds.

8.3 Conclusion

In this thesis I have presented an example of engineering complex global shape and pattern from local interactions. I present a language for instructing a sheet of identically-programmed, flexible, autonomous cells to assemble themselves into a predetermined global shape, using local interactions. The desired global shape is described as a folding construction on a continuous sheet, using a set of axioms from paper-folding (origami). I provide a means of automatically deriving the cell program, executed by all cells, from the global shape description. The cell program is inspired by developmental biology. With this language, a wide variety of global shapes and patterns can be described at an abstract level, and then synthesized using only local interactions between identically-programmed cells.

Bibliography

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43(5), May 2000.

[2] H. Abelson, G. Sussman, and J. Sussman. *The Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, MA, 1995.

[3] S. Adams. A high level simulator for gunk. Ai memo, MIT Artificial Intelligence Lab, November 1997.

[4] Y. Bar-Yam. *Dynamics of Complex Systems (Studies in Nonlinearity)*. Perseus Publishers, 1997.

[5] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH '98*, Orlando, FL, July 1998.

[6] J. Bard. *Morphogenesis*. Cambridge University Press, U.K., 1990.

[7] R. Beckers, O. Holland, and J. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. *Artificial Life*, 4, 1994.

[8] A. Berlin. *Towards Intelligent Structures: Active Control of Buckling*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, May 1994.

[9] M. Bern and B. Hayes. The complexity of flat origami. In *Proc of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175–183, Atlanta, January 1996.

[10] W. Butera and V. Bove. Literally embedded processors. In *Proc. of SPIE Media Processors*, 2001.

[11] P. Cheung, A. Berlin, D. Biegelsen, and W. Jackson. Batch fabrication of pneumatic valve arrays by combining mems with printed circuit board technology. In *Proc. of the Symposium on Micro-Mechanical Systems, ASME Intl. Mech. Engineering Congress and Exhibition*, Dallas, TX, November 1997.

[12] Web3D Consortium. Vrml repository. http://www.web3d.org/vrml/.

[13] D. Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, February 1999.

[14] D. Coore, R. Nagpal, and R. Weiss. Paradigms for structure in an amorphous computer. AI Memo 1614, MIT Artificial Intelligence Lab, 1997.

[15] E. Demaine, M. Demaine, and J. Mitchell. Folding flat silhouettes and wrapping polyhedral packages. In *Annual Symposium on Computational Geometry*, Miami Beach, Florida, 1999.

[16] J. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting: robot-like ants and ant-like robots. *Simulation of Adaptive Behavior: from animals to animats*, 1990.

[17] R. D'Souza. *Macroscopic order from reversible and stochastic lattice growth rules.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, September 1999.

[18] C. Folk and C-M. Ho. Micro-actuators for control of delta wing with sharp leading edge. In *39th AIAA Aerospace Sciences Meeting and Exhibit*, January 2001.

[19] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, 13:285–319, 1993.

[20] E. Frigerio and H. Huzita. A possible example of system expansion in origami geometry. In *First International Meeting of Origami Science and Technology*, Ferrara, Italy, 1989.

[21] T. Fukuda and S. Nakagawa. Approach to the dynamically reconfigurable robot systems. *Journal of Intelligent and Robotic Systems*, 1:55–72, 1988.

[22] T. Fuse. A folded deep pot. In *Third International Meeting of Origami Science Math and Education*, Asilomar, CA, March 2001.

[23] S. Hall, E. Crawley, J. Howe, and B. Ward. A hierarchic control architecture for intelligent structures. *Journal of Guidance, Control and Dynamics*, 14(3):503–512, 1991.

[24] C. Hanson. Mit scheme reference manual. AI-TR 1281, MIT Artificial Intelligence Lab, January 1991.

[25] J. Heath, P. Kuekes, G. Snider, and R. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, 1998.

[26] R. Hoffman. Airbag folding: Origami design applied to an engineering problem (easi engineering gmbh, germany). In *Third International Meeting of Origami Science Math and Education*, Asilomar, CA, March 2001.

[27] T. Hogg, H. Bojinov, and A. Casal. Multiagent control of self-reconfigurable robots. In *4th International Conference on Multi-Agent Systems (ICMAS)*, July 2000.

[28] B. Holldobler and E. Wilson. *The Ants*. The Belknap Press of Harvard University Press, Cambridge, MA, 1990.

[29] T. Hull. Origami mathematics website and combinatorial geometry class notes. http://web.merrimack.edu/∼thull/.

[30] H. Huzita. The trisection of a given angle solved by the geometry of origami. In *First International Meeting of Origami Science and Technology*, Ferrara, Italy, 1989.

[31] H. Huzita and B. Scimemi. The algebra of paper-folding. In *First International Meeting of Origami Science and Technology*, Ferrara, Italy, 1989.

[32] R. Hyde. Eyeglass 1: Very large apeture telescopes. *Applied Optics*, 38:4198–4212, 1999.

[33] R. Hyde and S. Dixit. Use of origami in fielding very large space telescopes. In *Third International Meeting of Origami Science Math and Education*, Asilomar, CA, March 2001.

[34] D. Ingber. The architecture of life. *Scientific American*, January 1998.

[35] R. Jackman, S. Brittain, A. Adams, M. Prentiss, and G. Whitesides. Design and fabrication of topologically complex, three-dimensional microstructures. *Science*, 280:2089–2091, 1998.

[36] A. Jacobson, G. Oster, G. Odell, and L. Cheng. Neurulation and the cortical tractor model for epithelial folding. *Journal of Embryology and Experimental Morphogenesis*, 96:19–49, 1986.

[37] K. Kasahara. *Origami Omnibus*. Japan Publications, Inc, Tokyo, 1999.

[38] T. Kawasaki. The geometry of orizuru. In *Third International Meeting of Origami Science Math and Education*, Asilomar, CA, March 2001.

[39] L. Kleinrock and J. Silvester. Optimum tranmission radii for packet radio networks or why six is a magic number. In *Proc. Natnl. Telecomm. Conf.*, pages 4.3.1–4.3.5, 1978.

[40] B. Kresling. Folded and unfolded nature. In *Origami Science and Art: Second International Meeting of Origami Science and Scientific Origami*, Otsu, Japan, 1994.

[41] R. J. Lang. A computational algorithm for origami design. In *Annual Symposium on Computational Geometry*, Philadelphia, PA, 1996.

[42] P. A. Lawrence. *The Making of a Fly: the Genetics of Animal Design.* Blackwell Science Ltd, Oxford, U.K., 1992.

[43] K. J. Lee, W. D. McCormick, Q. Ouyang, and H. L. Swinney. Pattern formation by interacting chemical fronts. *Science*, 261:192–194, July 1993.

[44] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Wonderland, 1996.

[45] J. Maekawa. Evolution of origami organisms. *Symmetry: Culture and Science*, 5(2):167–177, 1994.

[46] N. Margolus. Cam-8: A computer architecture based on cellular automata. *Pattern Formation and Lattice-Gas Automata, American Mathematical Society*, pages 167–187, 1996.

[47] M. Mataric. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16((2-4)):321–331, December 1995.

[48] M. Mitchell, J. Crutchfield, and P. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[49] M. Mitchell, J. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? *Advances in Neural Information Processing Systems*, pages 285–319, 1994.

[50] K. Miura. Concepts of deployable space structure. *International Journal of Space Structures*, 8(182):3–16, 1993.

[51] K. Miura. The application of origami science to map and atlas design. In *Third International Meeting of Origami Science Math and Education*, Asilomar, CA, March 2001.

[52] J. Montrol. *Origami Inside-Out.* Dover Publishers, 1993.

[53] R. Nagpal. Organizing a global coordinate system from local information on an amorphous computer. AI Memo 1666, MIT Artificial Intelligence Lab, 1999.

[54] R. Nagpal and D. Coore. An algorithm for group formation in an amorphous computer. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, October 1998.

[55] C. Nusslein-Volhard. Gradients that organize embryo development. *Scientific American*, August 1996.

[56] G. Odell, G. Oster, P. Alberch, and B. Burnside. The mechanical basis of morphogenesis: 1. epithelial folding and invagination. *Developmental Biology*, 85:446–462, 1981.

[57] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian. Useful metrics for modular robot planning. *IEEE Transactions on Robotics and Automation*, 13(4), August 1997.

[58] Pearson. Complex patterns in a simple system. *Science*, 261:189–192, July 1993.

[59] Philips, Shivendra, Panwar, and Tatami. Connectivity properties of a packet radio network model. *IEEE Transactions on Information Theory*, 35(5), September 1998.

[60] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5), May 1999.

[61] P. Doyle R. Lang. *Origami Insects and Their Kin.* Dover Publishers, 1995.

[62] M. Resnick. *Turtles, Termites and Traffic Jams.* MIT Press, Cambridge, MA, 1994.

[63] J. Slack. *From Egg to Embryo, second edition.* Cambridge University Press, U.K., 1991.

[64] D Arcy Thompson. *On Growth and Form, abridged edition.* Cambridge University Press, U.K., 1961.

[65] F. Val. Genetic analysis of the morphological differences between two interfertile species of hawaiin drosophila. *Evolution*, 31:611–629, September 1977.

[66] R. Scheaffer W. Mendenhall, D. Wackerly. *Mathematical Statistics with Applications.* PWS-Kent Publishing Company, Boston, 1989.

[67] R. Weiss, G. Homsy, and T. Knight. Toward in vivo digital circuits. In *Dimacs Workshop on Evolution as Computation*, January 1999.

[68] R. Weiss, G. Homsy, and R. Nagpal. Programming biological cells. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '98), Wild and Crazy Ideas Session*, October 1998.

[69] B. C. Williams and P. P. Nayak. Immobile robots: Ai in the new millennium. *AI Magazine*, 1996.

[70] A. Witkin and D. Baraff. Physically based modeling: Principles and practice. In *SIGGRAPH '97 Course notes*, 1997.

[71] L. Wolpert. Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology*, 25:1–47, 1969.

[72] L. Wolpert. *Principles of Development.* Oxford University Press, U.K., 1998.

[73] M. Yim. *Locomotion With A Unit-Modular Reconfigurable Robot.* PhD thesis, Stanford, Department of Computer Science, 1994.

[74] M. Yim, J. Lamping, E. Mao, and J. G. Chase. Rhombic dodecahedron shape for self-assembling robots. Spl techreport p9710777, Xerox PARC, 2000.