

Toward Scalable Morphogenetic Engineering: Natural Computing in SPH Swarm Control

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Allen Clark McBride

May 2022

Acknowledgments

Thanks to my advisor, Professor Bruce MacLennan. His guidance and mentorship have been indispensable, as have our clarifying conversations about emergence, embodied computation, and the nature of the discrete and continuous. Thanks also to my committee members, Professors Lynne Parker, James Plank, and Louis Gross, for their advice and guidance on the direction and presentation of my research. I am also grateful to the support and direction offered by my fellow lab members, especially Casey Miller, Dr. Zahra Mahoor, Aaron Mishtal, and Camille Crumpton. I have depended on my friends as well for their encouragements and helpful conversations, especially Rose Williams, Michael Sewell, and Tim Mowrer. I am especially indebted to my partner Jessie Pounds for her support, encouragement, and patience, as well as helpful conversations on the discrete and continuous. Finally, I am deeply grateful to my sister Bonny Domoto and to my father and stepmother Neil and Maureen McBride for their support, encouragement, and understanding.

Abstract

Artificial morphogenesis (or morphogenetic engineering) seeks inspiration from developmental biology to engineer self-organizing systems. The Morphgen language uses partial differential equations (PDEs) to express artificial morphogenetic processes as spatial fields describing large numbers of agents in the continuum limit. I present an approach to compile such systems of PDEs by discretizing their behavior to derive controllers for finite numbers of agents of finite size. This approach builds on a generalization of methods to control swarms of robots based on the computational fluid dynamics technique of smoothed particle hydrodynamics (SPH). I address potential scalability and efficiency challenges in SPH robotics by proposing embodied kernel functions that allow key operations to be offloaded to the physical environment. These kernel functions assume the ability of agents to produce local physical fields with certain properties. I especially explore the possibility of cell-like agents able to control the secretion rate of morphogens that diffuse and are degraded in an aqueous medium. I present strategies to mitigate gaps between simplifying assumptions and expected realities. I explore in simulation the validity, accuracy, and robustness of these strategies and of my overall approach.

Table of Contents

1	Introduction	1
1.1	Inspiration: Morphogenesis, Natural and Artificial	1
1.2	Objectives and Contributions	3
2	Background and Literature Review	6
2.1	Compiling the Morphgen Language	8
2.2	Continuous Descriptions of Biological Processes	10
2.3	Global-to-Local Compilation	11
2.3.1	Amorphous Computing and Global-to-Local Compilation	11
2.3.2	PDE Control in Swarm Robotics	14
2.3.3	Other Types of Global-to-Local Compilers	17
2.3.4	Global-to-Local Compilation and Emergence	18
2.4	SPH Robotics	19
2.4.1	Background: Smoothed Particle Hydrodynamics	19
2.4.2	Advantages of SPH Robotics	22
2.4.3	Closely Related Techniques	23
2.4.4	Examples of SPH Robotic Control	25
2.5	Agent-level Mechanisms	30
2.5.1	Robots vs. Synthetic Biology	30
2.5.2	Simulated Morphogens	32
2.5.3	Physical Morphogens	33
2.5.4	Molecular Communication	34
2.5.5	Electromagnetic Signal Strength	36

3	Theory	38
3.1	Natural Smoothing Functions	39
3.1.1	SPH Estimates	40
3.1.2	Spatial Derivatives	41
3.1.3	Signed and Vector Fields	43
3.1.4	Benefits of NSF Framework	44
3.2	Radiative and Evanescent Smoothing Functions	45
3.3	Morphogen Smoothing Functions	51
3.3.1	Idealized MSFs	51
3.3.2	Practical Considerations	54
3.4	Correcting for Transient Spurious Morphogen Smoothing Functions	57
3.4.1	Changes in Production Rates	58
3.4.2	Agent Motion	61
3.5	Beyond Fluid Dynamics: SPH as a Partial Morphogen Compiler	64
3.6	Spatial Mixtures of Properties	67
3.7	Growth and Apoptosis	70
3.7.1	Integral Control Approach	73
3.7.2	Proportional Control Approach	74
3.7.3	Interaction with Mixed Differentiation States	76
3.7.4	Control Theory	77
3.7.5	Reproduction By Recruitment	78
3.8	Edges	80
3.8.1	Mass Density Gradient	81
3.8.2	Color Fields	83
3.9	Upwind Scheme for Advection	86
3.10	Maintaining Smoothness	89
3.10.1	Smoothness of Mass Density Field	91
3.10.2	Smoothness of Velocity Field	94
3.10.3	Interaction with Mixed Differentiation States	96
3.11	Compilation	97
3.11.1	Stage I: Mapping Variables to Morphogens	98
3.11.2	Stage II: Implementing NSF-SPH	100

3.11.3	Stage III: Targeting Future Technology	103
4	Simulation Model	105
4.1	Overview of Control Flow	105
4.2	Overview of Agents	107
4.3	Nucleus	109
4.4	Extranuclear Agent Behavior	110
4.5	Agent Geometry	111
4.6	Sensors	113
4.7	Motion	116
4.8	Collisions	118
4.9	Hardware Failure	119
4.10	Reproduction and Apoptosis	120
4.11	Fluid Medium	121
4.11.1	Crawling	121
4.11.2	Swimming	122
5	Results from Simulation	124
5.1	A Simple Diffusion Test Program	125
5.2	Agent Size and Shape	128
5.3	Physical Morphogen Parameters	131
5.4	Agent Imperfections: Noise, Bias, Hardware Failure	133
5.4.1	Avoiding Spurious Results in Measuring Accuracy	133
5.4.2	Sensors	136
5.4.3	Motion	140
5.4.4	Hardware Failure	142
5.5	Agent Speed and Calibration	144
5.6	Correction for Laplacian Operator	148
5.7	Upwind Scheme	150
5.8	Demonstrating Growth	151
5.9	Integral and Proportional Control	158
5.10	Density Corrections for Recruitment	162
5.11	Demonstrating Mixed Differentiation States	164

5.12	Edge Finding	167
5.13	Path Finding	167
5.14	Crawling vs. Swimming	177
6	Concluding Discussion	182
6.1	Education	183
6.1.1	Speculative Classroom Implementation	183
6.1.2	Educational Benefits	185
6.2	Ethics	186
6.3	Future Work	188
6.4	Conclusion	190
	Bibliography	191
	Appendices	208
A	Computational Fluid Dynamics	209
A.1	Modeling Fluid Velocity	209
A.2	Modeling Diffusion	213
A.3	Stalks for Agents' Instruments	215
	Vita	218

List of Tables

3.1	Idealized radiation smoothing functions	48
3.2	Idealized morphogen smoothing functions	53
5.1	Default parameters for diffusion test program	127
5.2	Default parameters for growth program	157
5.3	Parameters for path finding program	172

List of Figures

2.1	Situating my approach at the intersection of three key areas of research.	7
3.1	Comparison of idealized MSF, RSF, and Gaussian smoothing functions in two and three dimensions.	55
3.2	Toy simulation to illustrate spatial vs. non-spatial aspects of non-steady-state conditions	59
3.3	Two colocated tissues, distinguished by differentiation states, moving through each other	69
3.4	Deriving expected edge gradient for half-plane grid of agents	82
3.5	An NSF-SPH agent's upwind derivative estimate in the context of needed assumptions	90
3.6	A pitfall of unsmooth density fields in estimating gradients.	92
4.1	Schematic of an agent in simulation	112
5.1	Demonstrating heat equation Morphgen program.	127
5.2	Accuracy of heat equation test program by smoothing length, for various agent diameters.	130
5.3	Accuracy of heat equation test program for different eccentricities of agents' elliptical body shapes.	132
5.4	Accuracy of the heat equation test program for different morphogen physical parameters.	134
5.5	Demonstrating heat equation Morphgen program with lateral motion.	137
5.6	Accuracy of heat equation test program for different sensor bias spread parameters. .	137
5.7	Accuracy of heat equation test program for different association constants for binding of morphogen to sensors.	139
5.8	Accuracy of heat equation test program by half-life of morphogen bound to sensors, for different free morphogen half-lives.	139

5.9	Accuracy of heat equation test program for different amounts of bias in agents' motion.	141
5.10	Accuracy of growth test program for different standard deviations for random walk noise.	143
5.11	Accuracy of heat equation test program for hardware mean times to failure.	143
5.12	Accuracy for heat equation test program by ratio of artificial speed multiplier to diffusivity.	145
5.13	Accuracy for heat equation test program by artificial speed multiplier.	147
5.14	Comparing paraboloid generation to ideal results for heated disk in bath, with and without correction for Laplacian operator.	149
5.15	Demonstrating advection Morphgen program.	152
5.16	Comparing pattern advection to ideal results, with and without upwind scheme.	152
5.17	Demonstrating stem growth program.	155
5.18	Robustness of stem growth program to cutting out agents.	156
5.19	Accuracy of growth test program, using proportional control, for different half-lives for the leaky integral used in this approach.	159
5.20	Comparing integral and proportional control approaches.	159
5.21	Example of tumor-like overgrowth in plant stem growth program.	161
5.22	Accuracy of growth test program, using integral control, with and without corrections for two distinct challenges for a recruitment model of reproduction.	163
5.23	Demonstration of mixtures with two crossing stem-like structures.	166
5.24	Comparison of approaches for edge finding.	168
5.25	Simulations of path-finding Morphgen program for a range of agent sizes.	173
5.25	(continued)	174
5.26	Accuracy of swimming and crawling models with two heat equation test programs.	178
5.27	Accuracy of swimming and crawling models with advection test programs.	178
5.28	Accuracy of swimming and crawling models with plant growth test programs.	180
5.29	Swimming vs. crawling in a path-finding simulation.	180
1	Motion of fluid in response to agent translation.	214
2	Assumed geometry of partially obstructed cells for diffusion boundary conditions.	216

List of Abbreviations

cAMP	Cyclic adenosine monophosphate (small signalling molecule that acts as a morphogen in <i>Dictyostelium</i>)
Dpp	Decapentaplegic (example of a large biological morphogen, best studied in <i>Drosophila melanogaster</i>)
EM	Electromagnetic (in the present context, a family of approaches for generating NSFs)
FDM	Finite difference method (grid-based numerical method in which derivatives are approximated by differences between values in nearby grid cells)
GPL	Growing Point Language (language describing pattern formation in amorphous computing)
GRN	Gene regulatory network (network of genes whose expression levels are mutually regulated by their products)
GTL	Global-to-local (pertaining to global-to-local compilation)
GTLC	Global-to-local compilation (compilation that translates descriptions of a system's global behavior into behavior for the system's component parts)
MC	Molecular communication (communication mediated by molecules physically traveling from source to target)
MEMS	Microelectromechanical system (microscale machines with electronic and mechanical components)
MML	Mechanica Modeling Language (language for modeling biological tissue and processes in continuous terms)

MSF	Morphogen smoothing function (SPH kernel formed by the concentration of a dilute substance diffused and degraded in an aqueous medium)
MTTF	Mean time to failure (expected time before irreparable system failure)
NSF	Natural smoothing function (SPH kernel implicit in some physical field)
OSL	Origami Shape Language (language for artificial morphogenesis describing shape formation in terms of series of folds in finite flat sheets)
PDE	Partial differential equation (in the present context, the basis for descriptions of behavior in the Morphgen language)
RSF	Radiative smoothing function (SPH kernel formed by the amount of radiant power impinging on a given point)
RSS	Received signal strength (of radio signals; can be used to estimate distance from source)
SPH	Smoothed particle hydrodynamics (originally a Lagrangian technique for fluid simulation, also repurposed for robotic control)
TOTA	Tuples On The Air (method in self-organization of mobile networks)
VNSA	Violent nonstate actors (e.g., terrorist groups, insurgencies, militias)

List of Attachments

Simulation of basic heat equation Morphgen program (diffdemo.mp4)

Simulation of heat equation Morphgen program with lateral motion (difftransdemo.mp4)

Simulation of Morphgen program holding pattern despite agent motion (advectdemo.mp4)

Simulation of plant stem Morphgen program (growthdemo.mp4)

Simulation showing tumor-like overgrowth in plant stem Morphgen program (overgrowth.mp4)

Simulation of Morphgen program with two crossing plant stems (growxdemo.mp4)

Simulation of Morphgen program for path finding (pathdemo.mp4)

Chapter 1

Introduction

The smallest autonomous robots are currently of centimeter scale. Advances in microelectromechanical (MEMS) and nanoelectromechanical systems, as well as in synthetic biology, are likely to produce smaller and cheaper autonomous agents in the coming decades. What use could we make of hundreds of thousands or more similar, autonomous agents at the microscale or smaller? We can look to multicellular life for inspiration: natural selection has led to an overwhelming diversity of ways that collections of cells can self-organize into complex and useful forms. However, roughly three billion years separates the first multicellular life and the appearance of the human brain. My work contributes to strategies for bypassing evolution to engineer complex systems that are at once self-organizing and designed.

1.1 Inspiration: Morphogenesis, Natural and Artificial

Morphogenetic engineering or *artificial morphogenesis* takes inspiration from biological morphogenesis in its efforts to induce simple components to self-organize into complex functional forms [75]. Biological morphogenesis itself is “the organization of the cells of the body into functional structures via coordinated cell growth, cell migration, and cell death” [36]. Pattern formation, the generation of spatial variation in cell behavior over a tissue, is closely intertwined with morphogenesis during development, with the two processes building on each other [e.g., 35, 114].

Morphogenesis creates structures with appealing properties that traditional manufacturing processes may be unable to replicate. Traditional design and engineering requires central processes to construct elaborate plans and execute them precisely. By contrast, natural organisms both design and arrange themselves as they develop, creating complex structures that are adapted

to environments similar to those in which they developed. These structures are often scalable by necessity: the human heart, for example, must function even as it changes shape and grows orders of magnitude in size. Biological structures can also be robust to variations that arise during development, whether environmental (phenotypic plasticity) or internal. For example, a tree may grow around an obstacle, or a brain may learn tasks despite variations in limb form or function. Morphogenesis can produce structures that show robustness over time as well, in the form of self-healing properties.

These beneficial properties may be related to the bottom-up, self-organizing nature of morphogenesis. Artificial morphogenesis is motivated in part by the desire to harness these properties in designed structures [75]. As Pascalie et al. [109] say, morphogenetic engineering “establishes a new object of research at the intersection between traditionally disconnected domains: it stresses the programmability of self-organization, underappreciated in complex systems science, and, conversely, the benefits of self-organization, which are underappreciated in engineering.”

Considered as a computational system, morphogenesis is characterized by embodiment, emergence, and continuity. It is embodied because the developing body uses its own physical presence and properties to compute further development, so that computation and physicality are inextricable. It is emergent (in a broad sense) because it is difficult or impossible to predict the course of development from DNA without simulation or instantiation of the system. Morphogenesis is also characterized by continuity in various respects. Concentrations of morphogens; positions, shapes, and sizes of cells; and their changes over time are all continuous. Even the discreteness found in the individuality of cells may be approximated at the macroscopic scale, for at least some purposes, as continuous tissue. On the other hand, some aspects of morphogenesis do seem inherently discrete. Especially relevant to my work are the discrete identities of morphogen types or of other chemical species used in signaling.

Morphogenesis relies on a diversity of mechanisms. Gilbert [36] identifies 13 morphogenetic processes and categorize them as involving epithelial (connected) or mesenchymal (unconnected) cells. Artificial morphogenesis can take as a research program the emulation of as many of these processes as possible, first through distinct projects and later through unification into more general frameworks. My work takes inspiration from mesenchymal tissue, and especially from the mechanisms of cell migration, and to a lesser extent of cell division and cell death.

Viewing the field as a whole, diversity in morphogenetic mechanisms is reflected by a diversity of morphogenetic engineering projects, which overlap other fields including swarm robotics, modular

robotics, amorphous computing, cellular automata, and synthetic biology. Bodies developed through morphogenetic engineering may take form through the birth and death of sessile elements, or through the rearrangement of motile ones. Design approaches also vary. Processes may be designed by hand in an ad hoc fashion to illustrate particular hypotheses or principles. Designs may also be generated by hand in more systematic ways, for example through global-to-local compilation (GTLC). Evolutionary algorithms are also popular for automatic design of processes to achieve well-defined goals. Finally, a spectrum exists from biological morphogenesis with limited human control to fully artificial systems [143].

Doursat et al. [22] provide a taxonomy of morphogenetic engineering, including additional examples, and Oh et al. [102] provide a more recent review of similar work. Doursat et al. categorize work in morphogenetic engineering into four categories: “constructing,” “coalescing,” “developing,” and “generating.” Under the taxonomy of Doursat et al., my work best fits under the category of “coalescing” in that it involves large numbers of agents moving to form shapes. In addition, my work involves patterning, and to a lesser extent growth, which are traits of the “developing” category.

Perhaps the most ambitious potential application of artificial morphogenesis is the self-organization of neuromorphic computers [60]. Another possibility is self-constructing buildings that adapt their design during constructing to accommodate local environmental variations such as obstacles, uneven surfaces, or sunlight availability. In medicine, groups of microagents may be useful in repairing wounds [90], attacking tumors [131], delivering drugs to specific parts of the body [31], or performing other surgical tasks [28]. For example, microagents might travel through the bloodstream to a target site before coalescing and working together to mechanically attack a tumor or rejoin torn tissue. In the farther future, the in vivo growth of devices to replace organ function may be an application especially suited to my own work, which focuses on microscale agents working in an aqueous environment.

1.2 Objectives and Contributions

My overarching objective is to improve and extend smoothed particle hydrodynamics (SPH) robotics and to reframe it as a tool in compiling code for artificial morphogenesis based on partial differential equations (PDEs). In pursuit of this goal, I attempt to tie together two strands of research, making contributions to each. Both bodies of work will be described in chapter 2, but I introduce them here to situate my objectives. The first strand of research is the language Morphgen, which describes

morphogenetic processes, real or artificial. Morphgen describes tissues in the continuum limit using evolutionary PDEs. An important benefit of such continuum descriptions is to facilitate the scalability of implementations. However, prior to the present work, no method had been described to compile Morphgen, or similar PDE descriptions, to code that could run on discrete agents.

The second strand of research is SPH robotics. SPH itself is a method in physics simulation used to discretize continuous phenomena, primarily fluids. It simulates the relevant physical equations by updating properties (such as mass, position, and energy) of material points called “particles.” Perkinson and Shafai [112] introduced the idea that particles could be reinterpreted as robots, leading to swarms that behave as fluids. Since then, researchers in several labs have explored SPH control in swarms of both simulated and real robots. One characteristic of these implementations is that robots must communicate individually with each of their neighbors, evaluating pairwise expressions based on relative position and individual properties. Another characteristic is that the control equations used have described behavior that is mostly physical and related to fluid dynamics.

Although typically used in the context of fluid dynamics, the mathematical basis of SPH can be applied to a wide class of evolutionary PDEs, even non-physical. My work takes advantage of this generality to use SPH robotics as a basis for compiling a subset of the Morphgen language to code for individual robots. By reframing SPH control as a compilation technique, I contribute to the development of Morphgen as a practical language and, more generally, to the utility of continuous approximations as an approach to artificial morphogenesis. Simultaneously, I contribute to the generality of SPH robotics, opening up possibilities of swarm behavior beyond fluid dynamics and even beyond physics-inspired dynamics.

My other major contribution addresses potential practical challenges with scaling the communication needed for SPH robotics to massive swarms of hundreds of thousands to trillions of agents, as would be necessary to create, for example, structures with brain-scale complexity. The pairwise communication assumed thus far by SPH swarm robotics requires agents to avoid interference with other pairwise links being made nearby in the swarm, a task likely to become increasingly difficult with scale. In addition, agents must have a means of determining when they have made contact with all of their neighbors within some radius. I introduce an embodied communication protocol for SPH robotics with the potential to work with, rather than against, the physical communication medium to avoid both of these problems. With this new protocol, agents no longer need a concept of having individual neighbors, instead needing only the ability to produce and sense the strength or intensity of a limited number of fields around their bodies. In addition to circumventing the

challenge of interference, my embodied method also promises better efficiency through its natural parallelism.

I propose two variants of my embodied communication method, one involving radiative communication and the other molecular communication (MC). The MC variant assumes each agent has the cell-like ability to sense, and to control the secretion of, morphogen-like chemicals in an aqueous environment. I focus on this MC variant because I expect it to be more energy efficient than the radiative variant, and because it more closely resembles communication in biological morphogenesis. Accordingly, I focus on the scenario of large numbers of agents working in an aqueous environment.

To summarize, the two key problems my work addresses are:

- No method has been described to compile Morphgen, or similar PDE descriptions, to code that could run on discrete agents.
- SPH robotics is expected to face computational and communication challenges as the number of agents increases to biological scale.

My two primary contributions, which correspond to these problems, are

- a compilation technique for PDE-based artificial morphogenetic programs, based on a generalization of SPH robotics, and
- an embodied communication protocol to simplify communication in SPH robotics and improve its scalability.

I make several secondary contributions in addition to these primary two. Some of these secondary contributions address challenges that arise in exploring the interaction of my compilation approach and embodied communication protocol. Others extend the descriptive power of the Morphgen language in ways compatible with my compilation and communication approaches. These secondary contributions are detailed, along with my primary contributions, in chapter 3.

In chapter 2 I review the diverse research threads that intersect in my project, with an emphasis on SPH robotics. As mentioned above, my research contributions are then developed in chapter 3. I test these ideas in a simulation model described in chapter 4. Chapter 5 presents results from these simulations. Finally, chapter 6 concludes with thoughts about pedagogical applications, ethical implications, and future work. Some parts of this dissertation draw from material I contributed to MacLennan and McBride [80].

Chapter 2

Background and Literature Review

In this chapter I review some background material on which my own work builds, as well as similar and related work. Section 2.1 introduces the Morphgen language and discusses the need for a compiler to translate Morphgen code into control rules for individual agents. Besides Morphgen itself, I briefly describe (in section 2.2) one other language to describe biological processes as spatially continuous. The following section, section 2.3, situates the challenge of compiling Morphgen as an instance of global-to-local compilation (GTLC) and reviews literature relating to this concept. The particular method I build on to create a global-to-local (GTL) compiler for Morphgen is SPH robotics; this and related techniques are described in section 2.4, along with brief background on SPH itself. The final section in this chapter deals with literature related to future implementations of agents capable of executing code generated by the Morphgen compilation approach I describe.

Three key areas of literature stand out as especially important to my compilation approach. Amorphous computing (section 2.3.1) is the source of the GTLC concept and is similar to my work in various respects. SPH robotics (section 2.4) is, as mentioned, the key technique that I build on for my compilation approach. And of the two versions I present of my embodied, naturally parallel communication approach, the version I focus on (involving chemicals or morphogens that diffuse and are degraded in an aqueous medium) is an example of molecular communication (MC) (section 2.5.4). As illustrated in fig. 2.1, my work can be seen as lying at the intersection of these three primary areas of research.

No systematic review of the field of artificial morphogenesis is attempted in this chapter. Even so, the reader may find conspicuous the dearth of citations related to evolutionary approaches, which represent a major branch of artificial morphogenesis [e.g., 11, 41, 130, 21]. Although evolutionary

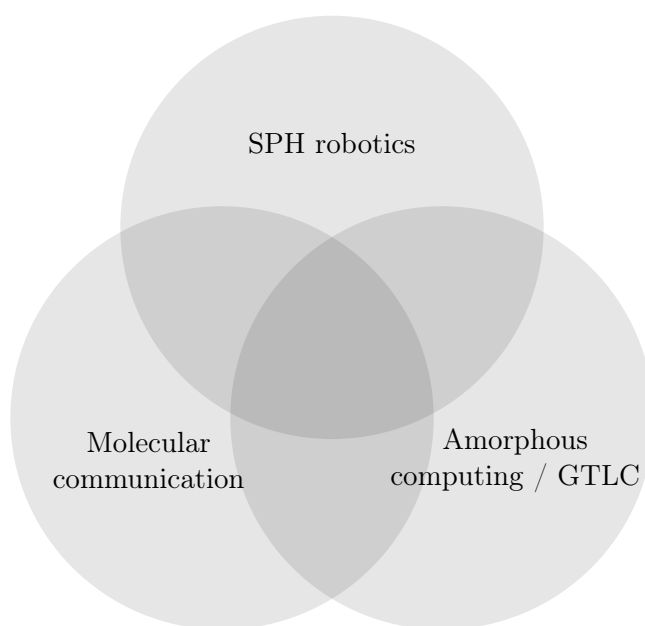


Figure 2.1: Situating my approach at the intersection of three key areas of research.

and other search-based optimization can yield impressive results, I focus on more direct compilation strategies. Hamann [40, p. 38] discusses further the limitations of evolutionary algorithms as a global-to-local design tool.

2.1 Compiling the Morphgen Language

The Morphgen language describes morphogenetic processes, biological or artificial [71, 72, 73, 74, 77, 76, 79, 80]. In both biological and artificial morphogenesis, processes are typically governed by finite numbers of finitely-sized agents, such as cells in epithelial tissue. However, Morphgen has no concept of individual agents or particles composing a substance. Instead, it is interested in the continuum limit of such processes, effectively treating tissue as being composed of infinite numbers of infinitesimally-sized agents. For this reason, Morphgen specifies behavior only in continuous terms in the form of PDEs.

Morphgen is object-oriented, with *substances* analogous to classes as abstractions with defined behavior and data structures, and *bodies* analogous to objects as instantiations of substances, with defined density and other fields. The behavior of a substance is composed of PDEs that may define a velocity or acceleration field, as well as other fields. These other fields may represent physical properties, differentiation state, or quantities useful for internal computations. The PDEs defining these fields are evolutionary, defining a time rate of change of a given quantity based on other variables and their (typically spatial) derivatives. The time rate of change operator is denoted “ $\dot{}$ ” to facilitate interpretation in discrete terms. Through PDEs, Morphgen can describe a wide range of processes including growth, motion, and patterning.

Morphgen substances can be active or passive, depending on the degree to which they can implement control processes and react to their environment. Active substances may be classified as physical or controllable, depending on the degree to which they can be programmed [80]. I focus on behavior specifications for active, controllable substances. Some substances may be controllable in the sense that their behavior can be hard-coded at manufacture time, and others may be reprogrammable in the field. My approach could apply to either scenario; the question of which is more feasible is a matter of future technology and economics.

These active, controllable substances are at the heart of artificial morphogenesis. As in biological morphogenesis, which is governed by collections of cells, in artificial morphogenesis these substances are typically composed of finite collections of discrete agents with finite size. Morphgen’s abstraction

of substances as continuous collections of infinitely many, infinitesimally small agents achieves one of its primary design goals, that of facilitating scalable implementation. By offering no facility for programmers to specify finite agent size or number, Morphgen ensures that its programs are, in principle, scalable. That is, a programmer cannot accidentally write a program that depends on agent size or number. Agents must be sufficiently small and numerous that the continuum limit can serve as an approximation for their collective behavior, but otherwise there is no conceptual limit to the range of agent size and number that can be used in an implementation.

On the other hand, we pay a price for the continuum abstraction that ensures scalability: it is not immediately clear how to compile a Morphgen program into code or specifications for the behavior of an individual agent. This is the challenge of GTLC: to take as input a global, collective description of system behavior and to output descriptions of the behavior of individual system components. In the case of Morphgen, an aspect of this challenge can be met by first translating, if necessary, PDEs to the Lagrangian¹ (material) frame so that time derivatives are with respect to moving agents and not to fixed points in space, as they are in the Eulerian frame [120]. Furthermore, the Lagrangian representation may be interpreted in terms of fluid parcels, which have some similarities with agents (for example, they can be assigned finite size and mass). However, fluid parcels cannot be interpreted as agents, because parcels must be treated as containing enough physical particles or agents that they have well-defined statistical properties such as density. To illustrate this point, consider that a typical Morphgen PDE may be a function of such variables as tissue density or spatial derivatives of certain fields (properties) of the tissue. While such quantities are well-defined over a continuous tissue, they have no obvious meaning with respect to an individual agent.

Therefore, further work is needed to generate desired behavior of individual agents. Specifically, a compiler should, for a given agent size (or, equivalently, number density), translate Morphgen’s tissue-level descriptions of behavior into specifications for individual agents. Rather than attempt to describe a comprehensive compiler for Morphgen (which has many features beyond those outlined here), I focus on a small but crucial subset of Morphgen. Specifically, I am concerned with the set of PDEs that describes the controllable behavior of an active tissue. Although Morphgen makes a useful point of reference, being an existing language, the ideas developed here would apply to similar behavior specifications. The essential element is the use of PDEs to describe the time evolution

¹Throughout this work, I use *Lagrangian* to mean that time derivatives are to be understood as from the perspective of a moving agent. That is, in the Lagrangian frame we are concerned with the Lagrangian or material derivative D/Dt , whereas in the Eulerian frame we are concerned with the partial derivative $\partial/\partial t$. But in both cases, spatial derivatives such as the gradient or Laplacian have their ordinary meaning in my work. I do not use Lagrangian descriptions in the fuller sense, in which spatial derivatives are taken with respect to particle labels.

of spatial fields of agents in the continuum limit. After the present chapter, the remainder of this dissertation describes and explores my method for this compilation.

2.2 Continuous Descriptions of Biological Processes

Morphgen is one instance of a language designed to describe biological processes in the continuous limit with respect to both space and time. Other such languages (as well as ideas not explicitly formalized as languages) are more relevant to the design of Morphgen itself than to the compilation approach described in the present work, although descriptions given in such languages may also serve as inspiration for Morphgen programs. I briefly describe one such language here because of its conceptual overlap with my own work.

The Mechanics Modeling Language (MML) describes biological tissue and processes [139]. It is designed to be compiled into code for simulating natural systems, which distinguishes it from my work, in which compiled code controls agents that emulate source descriptions. A commonality with my work is that MML emphasizes spatial considerations, even introducing spatial scoping of variables (which I do not explore). Like Morphgen, MML supports continuous modeling, both in time and space, although it also supports discrete modeling. Unifying continuous and discrete descriptions into one framework is one of the goals for MML; this contrasts with Morphgen, which is interested in larger spatial scales where tissue as well as growth processes can be approximated as continuous.

There are two other senses in which MML seeks to unify different sorts of descriptions into one modeling framework. First, MML addresses the challenge that different types of tissue, such as mesenchymal, epithelial, and fibrous tissues, might naturally be described by objects of different dimensionality. The authors seek novel data structures that can describe any of these tissue types [138]. Second, MML seeks to unify descriptions of active agents (primarily cells) with passive agents that might, for example, represent parcels of an aqueous medium. Morphgen also seeks to incorporate different types of materials (active vs. passive, programmable vs. non-programmable) into one modeling framework, although this aspect of Morphgen is not addressed in my work, which focuses on active, programmable tissue.

MML is also noteworthy because its authors mention the possibility of treating physical fields as approximation kernels, in the sense of SPH. They do not develop this idea, and their intent seems to diverge from mine, relating to calculating field values at locations away from locations where

the field value is known. They mention chemical concentration, though it is not clear whether they refer to diffusing and degrading chemicals. They also mention electric charge potential and give an expression for its kernel, though it decreases as the inverse of distance, so its integral over space would seem to diverge, disqualifying it as an SPH kernel.

2.3 Global-to-Local Compilation

GTLC has been defined formally by Yamins [163], but in the context of pattern generation among agents with fixed, regular topology, similar to cellular automata. Here I define GTLC informally, and more broadly, as an algorithmic process that takes as input a description of the overall behavior of a distributed multi-agent system and outputs descriptions of behavior for that system’s component agents from which the desired global behavior emerges. This definition is similar to that offered by Nagpal [99]. GTLC seeks to turn on its head the usual challenge of emergence: it seeks not to predict collective behavior from simple rules for individuals, but to derive such rules based on desired collective behavior.

2.3.1 Amorphous Computing and Global-to-Local Compilation

The idea of GTLC was introduced by Nagpal [96]. That dissertation and Nagpal [97] introduce a model of sheets of fixed-position agents that can pattern themselves and fold their sheets into three-dimensional shapes in the manner of origami. This model is inspired by the behavior of epithelial cells during morphogenesis, and is an example of amorphous computing. The source (global) language is the Origami Shape Language (OSL), which describes series of folds in finite flat sheets. The target (local) language describes the behavior of the individual agents.

Morphgen and OSL are both high-level, global languages in that neither has a concept of individual agents. OSL is higher level than Morphgen, in the sense that folding operations can be invoked without reference to the patterning and actuation processes that execute these folds. The OSL compiler works in part by translating these operations directly into agent-level subroutines. By contrast, Morphgen programs describe the formation of patterns in time and space at the global level. Another distinction is that Morphgen is in some ways more general than OSL: patterns are not restricted to plane Euclidean constructions, and tissues can move and deform in all available spatial dimensions.

At the agent level, the OSL compiler makes use of several agent-based primitives. Those most relevant to my work are gradient formation and neighbor query. The OSL compiler’s agent model assumes digital communication within a local neighborhood. Gradients are formed by broadcasting and incrementing integer values, a common mechanism in artificial morphogenesis and amorphous computing. Agents may also query their neighbors’ digital state, for example to determine the direction of steepest gradient ascent. My agent model is similar in its emphasis on local communication, but different in that communication is based on continuous values and is implicit, lacking the concept of individual neighbors (section 3.1.4). In addition, my model does not treat macro-level gradient formation as a primitive operation; instead, gradients are just one form of patterning that may be specified by a Morphogen program’s change equations. (The term “gradient” is overloaded; an agent’s estimate of the gradient *operator* with respect to a morphogen is indeed an important primitive in my model.)

Both Nagpal’s and my agent models attempt to judiciously include features on which the compilation method depends while abstracting away other features as implementation details. For example, the precise method of communication is left unspecified in Nagpal’s agent model, as are the mechanisms of actuation for folding and of internal computation. My model is more explicit about the medium of communication because it forms the basis for my compilation method. However, precise mechanisms of morphogen production and sensing (and also of propulsion and internal computation) are considered implementation details and are not modeled explicitly.

Nagpal elaborates on the idea of primitives in Nagpal [98], generalizing it to amorphous computation broadly. Although the primitives she catalogs still assume discrete communication between individual neighbors, these primitives are suggestive of strategies compatible with continuous descriptions and morphogen-based communication. In addition, Nagpal emphasizes connections between particular morphogenetic strategies and beneficial properties such as robustness and scalability. For example, morphogenetic processes may be more robust to variations in timing if sequencing depends not on absolute timers but on the completion of successive tasks.

Other work describes GTLC without using that term explicitly. (In fact, Doursat et al. [22] express the core challenge of all morphogenetic engineering in terms that could serve as a definition of GTLC: “How can the agents’ micro-rules be inferred from the system’s macro-objectives?”) This is true of Coore [17], who introduce the Growing Point Language (GPL) and its compilation, work that influenced OSL. GPL describes patterning of flat sheets via moving loci of control (a drawing process). It is implemented in simulation by agents embedded randomly and densely in a plane.

In this regard, the computational model is similar to OSL’s implementation. However, because the agents in Coore’s model do not move at all relative to each other, it may be considered a purer form of amorphous computing.

As in Nagpal’s work, the formation of virtual gradients is a key primitive operation at the micro level in Coore’s work. As is often the case in artificial morphogenesis, these gradients are virtual, implemented as values held by each agent and established through digital communication. This virtualness distinguishes Coore’s work and others’ from my own. However, my proposed physical morphogen fields echo some attributes that Coore emphasizes regarding his gradients. Specifically, Coore’s gradients are radially symmetric around the point of secretion, decrease monotonically with distance, and can be added together to form a summed field of gradients.

The examples of OSL and GPL are typical of amorphous computing in their focus on patterning across groups of agents with roughly uniform density. This clearly contrasts with my focus on motion and heterogeneous density of agents. However, patterning across groups of agents is also an important part of my work, whether for its own sake or to facilitate desired motion. To this end, methods in amorphous computing serve as inspiration for the patterning aspects of my own work.

The GTLC system of Kondacs [57] is also an example of amorphous computing, with agents unmoving but having the additional capability to reproduce or recruit new nearby agents. With this added ability, they use similar patterning tools as in the above work to form arbitrary, globally-specified two-dimensional shapes. Their compiler inscribes circles in the requested shape and generates code for agents to use gradients and leader election to reproduce wherever these circles are determined to be incompletely filled. Their work illustrates the power of GTL systems that allow for growth mechanisms, which play a role in my work as well (section 3.7).

Amorphous computing is extended to a model in which agents move randomly with respect to each other by the “Flying Amorphous Computer” [113]. This model may be considered an example of GTLC in the sense that the authors demonstrate the model’s ability to act as a universal digital computer. However, this is a less embodied sense of GTLC, because source programs would (as in traditional digital computing) be unrelated to the Flying Amorphous Computer’s spatial layout. The Flying Amorphous Computer nonetheless shares with my own work the ambition of expanding the range of computation that can be performed in an amorphous computing setting with agents that move arbitrarily with respect to each other.

The Proto language, like the Flying Amorphous Computer, extends the amorphous computing paradigm to include agents with arbitrary relative motion [3]. Unlike the Flying Amorphous

Computer, Proto allows for programmed motion, which brings it closer to my own goals. Although Proto and Morphgen both describe mobile agents in a continuum limit, Morphgen describes tissue behavior using PDEs, which ensures scalability. By contrast, Proto uses a more traditional programming approach of building up behavior from primitive operations. Individual primitive operations may or may not be scalable, at least in part because they may refer to individual agents, even though those agents are point-sized and embedded in a continuum of agents. For example, Proto can describe such non-scalable operations as assigning random headings to each agent or giving a special role to a single elected agent. Therefore, rather than a constraint imposed by the language as in Morphgen, scalability in Proto is a discipline practiced by the programmer through scalable composition of scalable primitives.

The ability to write non-scalable programs in Proto is one example of its generality with respect to Morphgen. As further illustration of its generality, we can consider that traditional SPH robotics (see section 2.4) could be specified in the Proto language. Proto allows the specification of arbitrary reduction functions over an agent’s neighbors, and SPH could be implemented by using a smoothing function for this reduction. It would therefore be possible to use many of the ideas in chapter 3 to compile Morphgen code to Proto code.

Proto, by design, is somewhat agnostic regarding the operations available in a given implementation, especially with respect to communication. Because Proto’s GTLC strategy is based on the composition of primitive operations, operations available at the global level depend on the primitive operations provided by a particular implementation. However, Proto generally assumes that agents will communicate individually with their neighbors, which contrasts with the natural parallelism of my embodied approach.

2.3.2 PDE Control in Swarm Robotics

Research describing the control of robot swarms using PDEs may be viewed as GTLC, because it involves the derivation of control laws for individual agents based on a global PDE description. Nonetheless, PDE control in the swarm robotics field has not typically been framed as GTLC. Because this section is the first focusing on swarm robotics as opposed to artificial morphogenesis, I begin with a brief introduction to swarm robotics.

Swarm robotics grew out of swarm intelligence and classical robotics [40], and often takes as its biological inspiration the collective behavior of groups of animals [129, 6, 12]. This perspective differs from artificial morphogenesis, which as previously noted takes its primary inspiration from

the development of individual organisms. Swarm robotics is often more focused on techniques that can be physically implemented or prototyped, whereas artificial morphogenesis is more likely to focus on simulations of techniques beyond the reach of current technology (Pascalie et al. [109] note the prevalence of simulation in artificial morphogenesis). Because both fields relate to emergent behavior from many similar agents, it is unsurprising that artificial morphogenesis often takes inspiration from swarm robotics, as is the case in the present work.

My work has some commonality with PDE swarm control methods that derive from control theory. (See Cao et al. [13] for a review of distributed multi-agent coordination systems from a control theory standpoint.) This body of research, as in my own work, describes desired swarm configurations using continuous approximations, and derives similar benefits in terms of scalability. Consistent with an influence from control theory, PDE swarm control typically focuses on continuous descriptions of static formations toward which a swarm can be made to converge in a provably stable fashion. It is these formation descriptions that are programmable, as opposed to the dynamics of convergence, which are generally described by a PDE that is fixed for a given control method. In contrast, I am interested in the challenge of programming dynamics with broad classes of PDEs, to which control theory analysis is less readily applicable. Nagpal [96] and Hamann [40, p. 28] address the limitations of control theory approaches to swarm robotics.

Several researchers have used methods based on PDE boundary control theory, in which control propagates recursively from agent to agent, often using the “backstepping” technique. These methods require one or more boundary agents to have more complete information about the overall state of the swarm than other agents, a leader-follower dynamic. Frihauf and Krstic [33] use this method to deploy groups of simulated agents to a family of planar curves in 2D space. Qi et al. [122] extend this work to simulate deployment to 2D manifolds in 3D space.

Both of these methods require only local communication, which is compatible with artificial morphogenesis. Otherwise, these boundary control methods are of limited relevance for my work. Although dynamics are described through PDEs, these PDEs are derived to drive agents to converge to particular families of manifolds. By contrast, Morphgen gives the programmer broad control over the dynamic behavior of tissue. Furthermore, these methods are not robust to agent failure or topological changes, which limits their relevance for tissues comprising large numbers of unreliable agents.

Krishnan and Martínez [62] use a related boundary control approach. Rather than targeting convergence to an arrangement on a given manifold, they target a given density function over a

separately defined and fixed two-dimensional area. They assume, without specifying a mechanism, that agents can sense boundaries. First, agents at the boundary use techniques of one-dimensional boundary control to form the desired boundary shape, after which they do not move. Agents in the interior then use this boundary shape as a reference in arranging themselves to match the desired density field. In addition to boundary sensing, it is assumed that agents can communicate individually with their neighbors and that they can sense their global orientation. Control laws are derived using a discrete Laplacian operator and kernel density estimation, which bears some resemblance to traditional SPH control.

Whereas the above control theory approaches tend to focus on convergence to a final arrangement, Berman and colleagues have described a family of related approaches involving PDE control of swarm dynamics [124, 9]. They model uncertainty in individual agent trajectories using the Fokker-Plank equation. This allows diffusion to be included in their macroscopic PDE model, based on the advection-diffusion-reaction equation. Applications include robotic pollination and mapping of unknown environments. Berman et al. focus on PDEs as descriptions of dynamics, which is compatible with the goal of compiling Morphgen code. Their work also illustrates an approach that Morphgen code can use to model state changes, which is to treat them as reactions that proceed quickly to saturation.

The clearest difference between the work of Berman et al. and my own is that they disallow any communication among agents, whereas I allow for communication through summed physical fields (which is still a more restricted assumption than traditional one-to-one or even one-to-many communication models). Another difference is that the macroscopic PDEs of Berman et al. relate to designed randomness in agents' motion, whereas my approach relates application-level PDEs to deterministic aspects of agents' motion. For example, consider the simple case of a dense cluster of agents programmed to execute simple Fickian diffusion at the macroscopic level. The method of Berman et al. would achieve this at the microscopic level via random motion of agents, whereas my method would instead use directed motion, with agents smoothly moving apart at the appropriate speeds (section 5.1).

Slavkov et al. [137] demonstrate PDE control of patterns within a swarm, based on agent state. They use the Kilobot platform to demonstrate patterning of swarms according to Turing pattern systems. The authors describe their control system as a gene regulatory network (GRN), although the analogy is loose and translation to biologically-plausible GRNs is not explored. Their focus on using PDEs to describe spatial patterns with no global coordinate system is similar to the goal of

Morphgen compilation. Communication among robots is digital, as in traditional SPH robotics, and requires locally unique IDs. PDE discretization for patterning involves simulated morphogen diffusion and a graph Laplacian, mechanisms loosely resembling a more digital version of SPH.

Slavkov et al. also explore interactions between Turing patterning and robot motion. In this regard their work bears little resemblance to the Morphgen approach or my partial implementation of it. Swarm dynamics are not described by PDEs but by an ad hoc algorithm meant to produce interesting and lifelike morphologies. Individual agents are not free to move as in my mesenchyme-inspired model; instead, agents move along the edge of a tightly-packed swarm. Finally, the authors demonstrate robustness to swarm damage. This robustness is consistent with other research that avoids fixed topology or global coordinates.

2.3.3 Other Types of Global-to-Local Compilers

Examples of GTLC outside the context of amorphous computing or PDE control tend to be less comparable to my work. One reason that some examples are less comparable is that they are highly digital in nature. That is, algorithms depend on such mechanisms as discrete state changes or message passing events. Furthermore, agents typically either start with fixed topological relationships with each other or form such relationships based on discrete connection potentialities.

Yamins [163] and Yamins and Nagpal [164] derive local rules for one- and two-dimensional cellular automata based on global pattern descriptions. Their model is precise and deterministic (other than random initial conditions), which facilitates analysis and exact solutions. Klavins et al. [56] work with robots able to attach to each other along edges depending on local state. They model assemblages of such robots as graphs, and use graph grammars to derive local attachment rules from global graph descriptions and transformations. Arbuckle and Requicha [2] also work with robots able to attach along edges. Their compiler takes polygons as input and generates local rules that use message loops to create and maintain connections. Generated structures have self-healing properties because structure maintenance is based on messages alone and not on internal state.

Werfel et al. [160] take inspiration from termite mounds to compile descriptions of three-dimensional structures into code for agents to build these structures by stacking blocks. Other than avoiding neighbors, agents do not communicate directly; instead, all communication is indirect through the growing structure. This communication through the environment makes their work a classic example of the use of stigmergy in swarm robotics.

My morphogen-based communication method can also be seen as an example of stigmergy, following the definition used by Werfel et al. of “indirect coordination through manipulation and sensing of a shared environment.” On the other hand, although theirs is a typical definition of stigmergy, in practice, “stigmergy” is usually applied to systems in which the temporal persistence of information in the environment is essential to the system’s function. By contrast, the temporal persistence of morphogen fields in my own approach is more a necessary evil than an essential element of the communication model. My communication method is therefore not clearly stigmergic in the classic sense of, for example, ant pheromone trails or termite mounds.

In some GTLC work, agents do not have explicit spatial positions, or are assumed to be well-mixed in their spatial domain. Typically such work focuses on task-switching or task-allocation challenges. Berman et al. [8] use differential equations to describe the evolution of proportions of agents engaged in different tasks. Valentini et al. [153] also derive task-switching rules from global information. A programmer can specify a desired sequence of probability distributions over possible allocations between two tasks. This sequence can be compiled into probabilistic rules that each agent can follow to periodically decide, based on its neighbors’ tasks, whether to switch its own task. Although I do not explore the implications of GTLC task-switching work here, in the future these techniques may be relevant to Morphgen programming in light of the approach in section 3.6 to describe mixtures of agents in different differentiation states.

2.3.4 Global-to-Local Compilation and Emergence

Hamann [40] offers a view of GTLC that contrasts with that of Nagpal described earlier. Hamann is skeptical that direct, algorithmic GTL programming is possible. Instead, he argues for an iterative approach of designing local rules, modeling their emergent collective properties, and using such modeling results to guide the design of improved local rules. In fact, Hamann goes so far as to say that GTL and local-to-global programming are fundamentally the same problem. I disagree, because the space of possible local rules may be so large that, even with efficient modeling of each ruleset, a search for a ruleset with desired global properties may not be feasible. Therefore I consider that GTLC is an important problem regardless of the state of the art in local-to-global modeling.

In light of his view that GTL and local-to-global programming are interchangeable, Hamann [40] focuses on techniques for modeling agent-based systems as continuous, PDE-based systems. This may at first appear to be the inverse of the goal of GTLC. However, because Hamann’s simulations are intended to be practical, yet more efficient, tests of agent systems’ efficacy, the global PDEs

would reflect not only the abstract intentions of the designer, but also practical considerations such as the behavior of fluid media and the effects of noise in agent behavior. One can even imagine a profitable compilation from a PDE-based language such as Morphgen, which would reflect abstract system goals, directly to a PDE-based simulation à la Hamann, which would attempt to model those practical concerns for which I use an agent-based model.

I am tempted to argue that Hamann’s skeptical position on algorithmic GTLC is refuted by the successful examples of the latter cited in this chapter. However, the different perspectives may be rooted in different views of what constitutes emergence and, by extension, the engineering of emergence via GTLC. Both Nagpal and Coore note that their compilation processes are straightforward, and I observe the same in my work. Thus the global behavior is, in at least two senses, unsurprising: Not only has it been programmed to begin with, but given an understanding of how the GTL compiler works, it can be inferred from the agent code. For this reason, Coore explicitly avoids the term “emergent behavior,” “to avoid the connotation of mystery in our outcome” [17]. However, I follow Nagpal’s lead in referring to the global behavior of the GTL compiled program as “emergent.” One definition of *emergent* that our group finds useful refers to global behavior that might, from some reasonable perspective (specifically, without knowledge of the global code or the compilation process), be surprising given agent behavior.

2.4 SPH Robotics

My compilation method for Morphgen is based on SPH robotics, which is in turn based on SPH itself. I briefly review each topic before turning to examples of SPH robotic control from the literature. My main purpose in reviewing the literature is to illuminate three themes that support the importance of my work. First, existing SPH robotics assumes pairwise communication schemes that may not scale well, a point elaborated in section 3.1.4. Second, existing literature fails to frame SPH robotics as an example of GTLC, a frame helpful in exploring the technique’s full potential. I discuss this reframing more in section 3.5 along with my third (related) theme, which is that existing literature primarily limits itself to deriving behavior that emulates fluid dynamics.

2.4.1 Background: Smoothed Particle Hydrodynamics

SPH itself is a meshfree Lagrangian numerical method used in physics simulation, primarily of fluid dynamics. It was introduced for modeling stars by Gingold and Monaghan [37] and Lucy [70]. It

is the most popular meshfree Lagrangian method for fluid dynamics simulations, and arguably the most straightforward. The chief advantage of SPH is its meshfree Lagrangian nature, which allows for simulation of large deformations without a separate method for making a new grid. Particles are updated according to their neighbors within a given radius at a given time, rather than according to their nearest neighbors defined by any fixed topology or mesh. So as particles' neighborhoods change, no extra work is needed to update the set of particles on which their future behavior will depend.

I briefly outline SPH, loosely following the textbook presentation of Liu and Liu [67]. The first step in deriving SPH is the observation that a function is unchanged by convolution with the Dirac delta function:

$$f(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}') \delta(\|\mathbf{x} - \mathbf{x}'\|) d\mathbf{x}' \quad (2.1)$$

If we replace the Dirac delta with a smoothing function, typically resembling a Gaussian, then we obtain a kernel approximation of f :

$$\langle f(\mathbf{x}) \rangle = \int_{\Omega} f(\mathbf{x}') W(\|\mathbf{x} - \mathbf{x}'\|, h) d\mathbf{x}'. \quad (2.2)$$

The symbol h stands for the *smoothing length*, which controls the spread of W such that

$$\lim_{h \rightarrow 0} W(\|\mathbf{x} - \mathbf{x}'\|, h) = \delta(\|\mathbf{x} - \mathbf{x}'\|).$$

The smoothing function's integral must be unity:

$$\int_{\Omega} W(\|\mathbf{x} - \mathbf{x}'\|, h) d\mathbf{x}' = 1.$$

Typically, the function is also required to be even, monotonically decreasing from the origin, positive, and reasonably smooth. However, various of these conditions have been relaxed in different applications, and other conditions beyond these may be imposed. Liu and Liu [67] analyze the accuracy implications of various smoothing function requirements.

Besides the above conditions, one more condition, that of compact support such that the smoothing function is strictly zero beyond some distance from the origin, is almost universally cited in the literature. However, this condition is of primarily computational importance. In SPH simulation and traditional SPH robotics, lack of compact support implies that pairwise calculations are needed between every pair of particles in the domain. For my embodied approach, a natural

parallelism circumvents this need for strictly compact support. Nonetheless, the requirement of a unity (and therefore convergent) integral implies a smoothing function that approaches zero quickly.

The next step in deriving SPH is to discretize eq. (2.2). $d\mathbf{x}'$ represents an infinitesimal volume element, and we can replace it by a finite volume element $\Delta V = m/\rho$, where m and ρ are the mass and mass density of the element, respectively. Focusing on a volume element located at position $\mathbf{x}$, and writing $W(r_j)$ as shorthand for $W(\|\mathbf{x} - \mathbf{x}_j\|, h)$, we can estimate $f(\mathbf{x})$ as

$$\langle f(\mathbf{x}) \rangle = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j),$$

where j indexes the neighbors of the focal volume element that are near enough so that $W(r_j) > 0$. These volume elements can be thought of as moving particles; each has definite position and carries mass and potentially other variables with it as it moves. (This is the sense in which SPH is a Lagrangian method.) It remains to track the changing density of each particle as it moves. The simplest method, which I adopt in my work, is to substitute $\rho(\mathbf{x})$ for $f(\mathbf{x})$ in the above, yielding

$$\langle \rho(\mathbf{x}) \rangle = \sum_j m_j W(r_j).$$

One approach to formulating SPH approximations of derivatives begins by substituting, say, $\nabla f(\mathbf{x})$ for $f(\mathbf{x})$ as the quantity to be approximated in eq. (2.1) [68, 67]. This approach requires Stokes' theorem and is not necessary for my purposes, being more important when smoothing length h varies in time or space. A simpler approach is to apply the derivative operator directly to the SPH interpolation and then to move it inside the summation [91, 92], for example:

$$\nabla \langle f(\mathbf{x}_i) \rangle = \nabla \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_{ij}) = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) \nabla W(r_{ij}).$$

Both approaches result in forms as above in which the derivative is applied directly to the smoothing function. Although more sophisticated forms of various SPH derivative approximations exist, this application of the derivative to the smoothing function is a defining feature of SPH. As explained in section 3.1.2, natural smoothing functions make it possible to circumvent the need to apply the derivative only to the smoothing function, instead measuring some derivatives directly in a physical interpolant field.

At this point, the next typical step in a presentation of SPH is to derive specific SPH approximations for fluid dynamics based on the Navier-Stokes equations or a simplified form such as the Euler equations. Indeed, most derivations of SPH robotic controllers would do so as well. I omit such derivations here: as mentioned in section 1.2 and discussed further in section 3.5, one of my central contributions is the application of SPH robotic control beyond fluid dynamics to a more general and potentially non-physical class of PDEs.

2.4.2 Advantages of SPH Robotics

Perkinson and Shafai [112] are the first to reimagine SPH as a means of swarm robotic control. Their key insight is that SPH particles can be reinterpreted as robots. Like robots, SPH particles have position and mass, and the rules used to update their state in a simulation can be reinterpreted as distributed control laws. Displacement becomes locomotion, and the pairwise evaluations of differences in position or other variables are computed by each agent communicating with its neighbors in turn.

SPH has several attractive features as a swarm control method. The compact support of SPH smoothing functions means communication can be strictly local. And needing no special effort to deal with even drastic changes in topology is helpful in swarm control, just as it is in physics simulation.

Scalability is also a key advantage. In physics simulation, adding more particles increases the resolution of the simulation while maintaining validity. Particle mass, of course, would be changed to hold density the same, reflecting the smaller volume represented by each particle. Similarly, in a swarm, either the number of robots or their mass can be changed. If they are changed reciprocally to hold density constant, no changes in the controlling PDEs are needed; otherwise, a simple variable substitution suffices. A similar substitution allows scaling of the collective swarm behavior independently of robot mass or number.

Another advantage, particular to the swarm application, is the smoothness of relative agent behavior. Because the behavior of SPH particles approximate solutions to PDEs, and because the solutions to PDEs must be differentiable with respect to those variables whose derivatives are described, the behavior of an agent under SPH control tends to be similar to its neighbors—not just statistically, but instantaneously. By contrast, some swarm control methods produce behavior that is not locally smooth, even though macroscopic behavior may solve a PDE (see the work of Berman et al. reviewed in section 2.3.2). Compared to such methods, the locally smooth behavior

of SPH control is more efficient and less likely to result in collisions. This advantage is noted by Eren and AıkmeŖe [25] in a context similar to SPH robotics.

2.4.3 Closely Related Techniques

SPH robotics was anteceded by Shimizu et al. [135], who derive decentralized control laws by similarly reframing physics simulation techniques as control laws to cause swarm behavior to resemble the simulated physical systems. Their work derives from Molecular Dynamics and Stokesian Dynamics rather than from SPH. Like SPH, these simulation methods are examples of meshfree particle methods. However, they lack the smoothing kernel approach that is central to SPH and which allows for the embodied variant introduced in my work. Stokesian and Molecular Dynamics are also more narrow than SPH in the equations they can approximate, because particles in these methods represent discrete physical entities rather than being discretizations of a continuum as in SPH.

As with the work of Shimizu et al., *physicomimetics*, or *artificial physics*, is similar to SPH robotic control in that it derives rules for individual agents to follow such that the system’s collective behavior will resemble physical systems [142]. Again as with Shimizu et al., *physicomimetics* does not use a continuum approximation; rather, each agent is more closely analogous to a physical particle, a molecule. Each agent accelerates based on the sum of forces with its neighbors, with each force calculation based on distance: repulsive for nearer neighbors, and optionally attractive for farther neighbors. These rules can generate behavior similar to solids, liquids, and gases. As with traditional SPH robotics, agents must be able to sense pairwise distance and heading with each neighbor, although only some applications require further information exchange.

Spears et al. [142] are interested in moving beyond what is physically plausible; for example, they discuss varying attractive forces in response to environmental cues so that, for example, a swarm can mostly behave as a solid but can become more liquid when moving through a field of obstacles. This flexibility is an important step toward the more complete freedom from physical behavior that my approach offers. Spears and Spears [141] further develop the idea of departing from strictly physical behavior, while remaining grounded in physics. This book also generalizes the definition of *physicomimetics* to include any control strategy designed to generate behavior inspired by physics. Traditional SPH robotics would be a special case of *physicomimetics* under this definition, although my work would not, as it seeks to move beyond any necessary link to physical behavior. Finally,

Spears and Spears [141] present a sample of various other developments in physicomimetics, which has become a popular swarm control technique.

In 2006, Muhammad Pac et al., seemingly unaware of the then-recent SPH robotics work of Perkinson and Shafai [112], proposed a meshfree Lagrangian discretization for the Euler equations (a special case of the Navier-Stokes equations) as a means of control for mobile sensor networks [105, 106]. This approach is similar in both concept and application to SPH robotic control. The discretization seems less standard than SPH, and it does not involve a smoothing function, so it would be less amenable to my embodied approach. Regardless, in later work the same authors abandon this early approach in favor of SPH control [107, 104].

Fujiwara et al. [34] simulate a control scheme that is reminiscent of my work. Like me, they are interested in agents communicating via their physical environment. Specifically, their agent communication is mostly (though not entirely) mediated by hydrodynamic forces, inspired by mechanisms used by schooling fish. To the extent that communication is mediated thus, their agents, like mine, do not need any concept of having individual neighbors. Furthermore, the authors similarly model each agent as a particle in an SPH scheme. On the other hand, communication via hydrodynamic force does not closely resemble communication via diffusing morphogens. In addition, SPH in Fujiwara et al. is best seen as part of their simulation technique, not as a source for deriving control laws as it is in most work combining SPH and swarm robotics. Fujiwara et al. use SPH mostly to model interactions between agents and parts of their fluid environment, so it appears that SPH would not be involved if their approach were physically realized.

Artificial potential field methods of swarm control can sometimes resemble SPH control in limited ways. Although early artificial potential fields were fixed, some authors have attached artificial potential fields to each agent in a swarm. The contribution of these fields to agent motion is typically limited to gradient descent, as compared to the more general value and derivative estimations seen in SPH. Hsieh et al. [45] present an example using artificial potential functions that decrease monotonically to zero with distance. Especially with such a potential field shape, summing gradients of potential field contributions from neighboring agents is similar to an SPH gradient estimate.

Some recent work has isolated the kernel density estimation part of SPH, using it as part of control laws that can bring robot swarms into agreement with predefined density fields [25, 165, 26, 27]. This research is not directly applicable to my own challenge, because it assumes that agents know their global positions, which they need to compare a local density estimate with a target density. However, in one respect this work is more closely related to my own than are the

approaches I identify in the next section as SPH robotics proper. This is because it abstracts at least one aspect of SPH away from its fluid dynamics origins to describe behavior that, although derived from a heat equation, bears little resemblance to a physical process. One example of such non-physicality is the use of single-integrator dynamics; all other SPH robotics uses the double-integrator dynamics consistent with Newton’s second law. Apart from this commonality with my work, the density estimation used in this work does not capture the potential of SPH to estimate derivatives of arbitrary spatial fields.

2.4.4 Examples of SPH Robotic Control

SPH was first proposed as a method for controlling robot swarms by Perkinson and Shafai [112]. Their work established precedents that would be followed by most other research in SPH robotic control. Their SPH equations follow the Navier-Stokes equations with a term for artificial, rather than physical, viscosity. Equations of state are chosen from those typically used in physics simulation for gases and liquids. Communication is assumed to be pairwise, and the approach is tested in simulation. It is assumed that agents can directly detect obstacles; the means of avoidance is not made clear, and authors since have explored various approaches.

Unusual for SPH robotic control, this work retains the energy part of the Navier-Stokes equations. It is not clear what benefit this has for typical robotic control applications, because density and momentum dominate temperature in determining the bulk motion of liquids. Most research in SPH robotic control focuses on the mass continuity and momentum equations of the Navier-Stokes or Euler equations. Also somewhat unusual, and notable because of its similarity to my work, a peaked kernel function is used; however, the peak is not so sharp as in natural smoothing functions. Finally, agents lower their pressure when they detect “objects of interest,” which causes other agents to gather around these objects. This approach is similar in spirit to the more traditional SPH virtual particle approach that can be used to cause agents to avoid obstacles.

Building on Perkinson and Shafai [112], Muhammad Pac and colleagues make advances in SPH robotic control, which they demonstrate in simulation [107, 104]. They introduce use of the full Navier-Stokes equations instead of the simplified Euler equations, which allows for the use of physical, rather than artificial, viscosity. (Both types of viscosity are, of course, artificial in that they are being emulated with a robot swarm; “physical” in this context refers to derivation from a model of physical viscosity.) They point out that temperature does not typically have a

major dynamic effect on the bulk motion of liquids, so that the energy-related part of the Navier-Stokes equations can be neglected or repurposed. They choose to repurpose it to facilitate obstacle avoidance, under the assumption that robots can sense obstacles directly. Most later researchers in SPH robotic control have chosen to neglect the energy equation entirely.

A second contribution of Pac et al. is to emphasize the usefulness of varying a range of fluid parameters without regard for physicality. They even discuss the possibility that parameters could be varied dynamically, perhaps in response to environmental cues. A higher-level layer of control for variation of parameters is proposed. This freedom represents a step away from being tied to fluid dynamics-related behavior, although they do not propose abandoning the form of the Navier-Stokes equations entirely. A more complete freedom from physical behavior is one of the contributions of my work. My approach also does not require a distinct level of control for parameter variation; instead, parameters may freely be replaced by functions in the underlying equations. Pac also mentions the possibility of heterogeneous swarms of robots, but does not develop an approach to this challenge, which is one of my goals [104].

Umut Tilki, along with Muhammad Pac’s coauthors Ismet Erkmen and Aydan Erkmen, follow up on Pac’s work by controlling simulated agents viewed as composing robotic hands or torsos [147, 148, 149, 150, 146]. Tilki et al. are interested in passive imitation learning by agents whose physiology is not comparable to the imitator. They adopt the two-level control structure of Pac et al. The higher-level controller is centralized, and uses an ANN and other tools to transform images of human hands or bodies into different parameters for the lower-level SPH controllers of each agent such that the collection of agents adopts a shape similar to the image. The parameters controlled are (physical) viscosity, stiffness, and a body force function, all of which are computed for distinct regions of agents. A given region might represent, for example, one joint of a finger.

In the context of my work, the most important contribution made by Tilki et al. is the idea of varying fluid parameters separately for different agents. This variation further escapes the confinement of mimicking physical behavior; as a result, the collection of agents as a whole cannot be said to behave as any single fluid substance could. On the other hand, all agents in their approach are still locally obeying fluid-like laws, which contrasts with my approach of allowing descriptions completely unrelated to fluid dynamics. Another limitation of this research is the implicit need for a central controller to communicate chosen parameters to each individual agent. This limitation is most serious with respect to body forces. These forces vary throughout the swarm according to

global location, which appears to implicitly require agents to have a means of knowing their global location.

Pimenta et al. [116, 115, 117] use SPH control in simulations and experiments to guide robot swarms toward goal formations. Goals are defined by the minima of electrostatic fields, and a term related to the gradient of this field is added to a term derived from SPH. Obstacles are avoided through the virtual particle method in SPH. As is typical of SPH robotic control, SPH equations follow the mass and momentum Navier-Stokes equations with the addition of artificial viscosity. The smoothing kernel is a typical cubic spline. Methods are simulated and also tested using Scarab and e-puck robots.

The authors demonstrate the scalability potential of SPH control more explicitly than other studies have, obtaining similar results in simulation using 81 and 1600 robots. This focus on scalability is similar to my work. On the other hand, their method requires a global localization system, which differs from my emphasis on fully decentralized control. One reason these studies use global localization is so that robots know where they are with respect to an emulated electrostatic field, but this need could be eliminated if a real electrostatic field were used, which agents could sense locally. Another reason for global localization is that it is needed to implement virtual SPH particles, which is a more serious limitation. I am not aware of a fully general and decentralized way to implement virtual particles, so I avoid them. The authors also pay special attention to control-theoretic analysis of convergence and collision avoidance. This also contrasts with my work, in which I am focused on a general method for compiling PDE behavior without regard for that behavior’s stability or convergence properties.

The use of an external electrostatic field for guidance also contrasts with my work. I include value and gradient estimates of both environmental fields and tissue-focused fields in the same source PDE descriptions. Only quantities relating to tissue-focused fields (such as density or more abstract fields) need to be compiled with SPH-related techniques, but at a mathematical level no distinction needs to be made between these and environmental fields.

Kamran Mohseni and colleagues explore the use of SPH in controlling unmanned aerial vehicles used for environmental monitoring [64, 133, 134, 65, 44, 111, 140, 136]. One key interest of this research is offline path planning for fuel optimization in the context of strong background flows of air (or, potentially, ocean water). Another is the real-time use of the swarm’s environmental sensor readings to steer agents to locations where further data collection can more efficiently contribute to knowledge of, for example, the size and shape of a pollutant plume.

The authors use SPH for swarm cohesion, collision avoidance, and guidance. As is typical in SPH robotics, they use SPH formulations of simplified Navier-Stokes continuity and momentum equations. They use a Murnaghan-Tait equation of state, and they sometimes include a term for artificial viscosity. SPH-based guidance is achieved through the use of virtual SPH particles with negative pressure to which other agents are attracted. Although swarm cohesion and collision avoidance can clearly be achieved through fully distributed SPH, the authors do not address how virtual SPH particles can be implemented without centralized guidance or localization. The authors present data based on small groups of real UAVs with varying capabilities, including GPS localization.

Aspects of some of these papers are relevant for my work. Song et al. [140] note that peaked smoothing functions seem to work well for collision avoidance, which is relevant because the natural smoothing functions explored in chapter 3 are peaked. Silic et al. [136] introduce asymmetric smoothing functions to correct for antenna that produce anisotropic signal strength. Such smoothing functions are an important contribution to making traditional SPH robotics practical for implementation on realistic hardware. However, they are not an option when natural smoothing functions are used, because the needed calculations for asymmetric kernels are inherently pairwise. This paper also introduces a “meticulous” scheduling scheme to ensure that only one agent is broadcasting to its neighbors at a given time. This scheme addresses communication latency, but its need also highlights the scalability challenges of communication in traditional SPH. Specifically, the requirement that agents receive communications separately from each neighbor imposes practical challenges that can be circumvented by my natural smoothing function method.

Argel Bandala, Elmer Dadios and colleagues use SPH in combination with body forces and virtual containers to control swarm behavior [4, 5, 83]. The authors define shapes such as cubes and spheres, and agents reverse direction when encountering the boundary of one of these shapes. Attractive points are simulated using body forces and are places beyond a boundary from the swarm. As a result, swarms congregate at controlled densities around the boundary nearest the attractive point. Experiments are carried out with real UAVs as well as in simulation; however, the SPH algorithm is controlled centrally. As with any SPH robotics approach that involves virtual objects located in absolute space with respect to the swarm, it is not clear how this approach could be made fully decentralized without agents having global localization capabilities.

Unlike most SPH robotics implementations, these authors use different smoothing functions for different purposes. This technique is relevant to my work, because each distinct physical morphogen

will have different diffusion and degradation characteristics, which implies different smoothing functions within the same SPH discretization. In addition, one of the smoothing functions they use has a singularity at the origin, which is unique among SPH robotics research. This singularity is also a commonality with my work, because some natural smoothing functions have singularities at the origin. Another unique point among SPH robotics work is the use of color fields (actually SPH approximations of unity) to implement surface tension. Color fields hold promise as a means not only to implement surface tension but also generally to define the boundaries of a swarm or tissue, which I explore in section 3.8.2.

Zhao [167] and Zhao et al. [168] explore combining SPH control with other types of swarm control. Their main interest in using SPH control is to control density, and they note that a physical emulation of fluids is not necessary to this end. Toward this goal, they simplify the SPH momentum equation to ignore neighbors' pressure and density, so that the only SPH calculation remaining is density and its gradient for the focal agent. They do not connect this simplified SPH equation with the PDE to which it would correspond, but their simplification is still notable for its departure from physical behavior. Four Khepera robots demonstrate formation of a diamond shape under their hybrid control scheme. In simulation, the authors also explore, but do not offer a full explanation of, spontaneous segregation of the swarm into separate circles.

Khaldi et al., drawing on the above work, incorporate SPH density estimates into a distance-weighted k-nearest neighbors approach to swarm aggregation [52, 53]. They test their approach in simulation. Other than the use of SPH to make local density estimates, this work does not draw on SPH. In a sense, the Khaldi et al. approach is therefore less related to my work than other research reviewed in this section, and is more directly related to k-nearest neighbor variants and other aggregation techniques. However, in another sense, their approach has a key commonality with my own: it is one of the few multi-agent control schemes to explicitly abstract the ability of SPH to provide certain estimates independently of any concern for fluid-like behavior. However, like the work reviewed at the end of section 2.4.3, it makes only partial use of the promise of SPH in this regard, using it only for density estimation. Like Zhao et al. mentioned above, Khaldi et al. do not present SPH estimates in the context of corresponding PDEs.

Powers [119] use SPH as part of an algorithm for escorting military assets, comparing it to an artificial potential field approach and finding advantages and disadvantages. Their overall algorithm is successful in simulation, but has several non-SPH components, limiting its general applicability to other SPH applications. Although their simulations contain relatively small numbers

of robots, they note that SPH seems to become more successful as agent number increases, which reinforces my interest in SPH as a control method for tissues comprising hundreds of thousands to trillions of agents. Furthermore, they explicitly note that as number of agents increases, individual-based communication is likely to become unreliable as well as inefficient in both time and power consumption. They observe that communication using chemical pheromones would address this challenge, but they argue that such communication is not sufficient to implement SPH control. One of my central contributions is to propose a solution to exactly this challenge: a means of using pheromone-like (more precisely, morphogen-like) chemical communication to implement SPH control, for better reliability and efficiency at scale.

2.5 Agent-level Mechanisms

My work is primarily intended for controlling large numbers of autonomous micro-scale agents in an aqueous environment. It is predicated on the idea that low-cost, microscale, autonomous agents are likely to be developed in the coming decades. Currently, fully autonomous agents are of centimeter scale at their smallest [e.g., 47, 127]. This limitation makes it difficult to predict the characteristics of suitable future technology. Therefore my work attempts to remain implementation-agnostic, targeting an agent model sufficiently abstract that it can apply to a range of possible future technologies.

In this section I briefly speculate about possible advantages of fully artificial and synthetic biological implementations. I then review a sample of literature that relates to my proposed models of agent communication, because these models are the basis for one of the central contributions made by my work. I first look at literature relating to the idea of agents communicating by simulated morphogen diffusion. Then I examine literature relating to communication through physical chemicals of varying similarity to morphogens. Finally I look at studies involving agents estimating distance by electromagnetic signal strength, which is one basis for my less-elaborated radiation-based alternative to morphogen communication. This review is not essential to an understanding of my work, because my communication models are explained in chapter 3.

2.5.1 Robots vs. Synthetic Biology

Microrobots based on microelectromechanical systems (MEMS) are one possibility for implementing my proposed model. Compared to synthetic biology implementations, artificial robots may offer the

advantage of more easily supporting arbitrary relationships between sensed gradients and headings. More broadly, as completely artificial entities, we can imagine the possibility of more precise control over all aspects of operation, including sensing, control, and computation.

Synthetic morphogenesis offers another possible route for implementation. Synthetic morphogenesis is a branch of synthetic biology in which control is exerted over the self-organization of collections of biological cells [143]. The collections may be of cells such as bacteria or protists that would not ordinarily undergo morphogenesis [109]. More likely they are the cells of multicellular organisms whose natural morphogenesis is modified. Control in synthetic morphogenesis may be exerted through modifications to the cells' GRNs with the assistance of designed environmental cues such as scaffolding and lights.

Because biological morphogenesis is the inspiration for artificial morphogenesis, synthetic morphogenesis is broadly attractive for implementing ideas in artificial morphogenesis. Metabolism, mitosis (division), and apoptosis (programmed death) are examples of cellular functions that could help implement methods in artificial morphogenesis. The framework presented in my work may be particularly well-suited to a synthetic morphogenesis implementation. My agent model is cell-like in that internal computation is a system of ODEs, which is considered a good model for GRNs [118]. Also, in the morphogen-based variant of my model, both secretion and sensing are idealizations of their biological counterparts, which further suggests the suitability of a synthetic morphogenesis implementation. Finally, in the context of medicine, a synthetic biology approach raises the possibility of meeting biocompatibility challenges by reprogramming a patient's own cells [28].

Challenges for a synthetic morphogenesis implementation will include cells being able not just to move up or down environmental gradients, but to move according to arbitrary vectors calculated internally with respect to those gradients. Compared to prokaryotic cells, eukaryotic cells typically can sense gradients more accurately and follow them more directly. Prokaryotic cells often follow a run-and-tumble strategy, randomizing their direction at a rate inversely related to how rapidly a relevant value is changing. Although less accurate than the more direct eukaryotic approach, such a strategy suggests the possibility that the value whose change is being tracked could be internally calculated rather than directly sensed.

2.5.2 Simulated Morphogens

A common approach in artificial morphogenesis and related fields has been to take inspiration from morphogen-based communication in biological cells, but to treat the idea as a metaphor. Rather than proposing that agents secrete chemicals into their environment, such approaches use more traditional communication technology to simulate morphogen diffusion. Some of these simulated diffusion strategies are more physically plausible than others.

Mamei et al. [81] were inspired by morphogenesis to explore the use of simulated gradients in simulated swarm self-organization. Robots communicate by broadcasting digital messages containing an integer for each simulated morphogen. The smallest number a given robot receives is incremented and re-broadcast. The resulting distributions of values therefore tend to approximate fields with constant slope, unlike physically diffusing morphogens. This mechanism is similar to mechanisms used in Nagpal’s OSL (section 2.3.1) but applied to swarms with locomotion.

Mamei et al. exploit gradients produced in this way in various digital algorithms to self-assemble shapes. The use of broadcast messages reduces but does not eliminate the need for robots to be individually aware of their neighbors. Most importantly, robots do not need to know their relative position with respect to each neighbor. This represents a step toward my own approach, which fully eliminates the need for robots to be aware of discrete neighbors.

Mamei et al. eliminate the need for robots to have directional sensing; instead, robots sense gradient direction by tracking values over time while wandering randomly. This mechanism is somewhat like run-and-tumble methods used by bacteria. This idea could be explored in conjunction with my approach, though I do not attempt it at present. Finally, Mamei et al. do not explore scalability in the sense that I am interested in, which is the ability of algorithm behavior to scale with swarm size.

Other researchers have used similar digital schemes, conceptually based on morphogen diffusion but not obeying the physical diffusion equation. For example, Rubenstein et al. [128] use such incremental gradients in a closely-packed Kilobot swarm, along with four pre-positioned “seed” robots, to establish a coordinate system. This coordinate system is then combined with edge-following behavior (as in Slavkov et al. [137]) to recreate bitmapped images. As a second example, Oh et al. [103] use incremental gradients in Kilobot swarms to explore pattern formation and swarm tracking and herding algorithms. This discrete, incremental gradient mechanism shares with my work an inspiration in morphogen diffusion. However, the abstract and non-physical nature of this

simulated diffusion limits its applicability to my work in proposing the use of physical morphogens for communication.

Some researchers have used simulated morphogen diffusion based on digitally communicated but continuously valued morphogen levels. Ramezan Shirazi et al. [125] simulate a distance-based method for agents to communicate morphogen levels to neighbors. This distance-based function brings their work closer to the behavior of physical diffusion, but is still distinct. Their work also avoids the need for agents to have directional sensing; as with Mamei et al. [81], agents use time-based gradient sensing.

The Tuples On The Air (TOTA) infrastructure for mobile networks [82] is similar to the above work in its communication mechanisms, but further explores the idea of abstract fields. This exploration represents an additional similarity to my work. In TOTA, distributed agents are conceptually situated and embodied, and use shared fields to self-organize. These shared fields may be produced by the combination in a virtual environment of each agent’s contribution to that field. Each agent’s contribution is shaped spatially by an arbitrary function of hop-count distance. In TOTA, this environment is an abstraction layer atop more or less conventional hardware. By contrast, my work proposes to take these ideas more literally, with methods befitting hardware able to create physical shared fields in their actual environment.

2.5.3 Physical Morphogens

Some artificial morphogenesis work has, as with mine, proposed the use of literal, morphogen-like chemicals as part of their approach. Pascalie et al. [109] describe the SynBioTIC project, in which their declarative SBGP language is compiled into simulated bacterial behavior. Like the Proto language described in section 2.3.1, their GTLC approach is based on modular algorithms that can be recombined into morphogenetic programs. Unlike Proto, these programs specify morphogens with program-specific diffusion and degradation rates. Their model of morphogen physical behavior, with Fickian diffusion and uniform, exponential decay, is identical to mine. However, the relationship between these morphogens and the program is typical of existing artificial morphogenesis work making use of literal morphogens. Specifically, the physical behavior (diffusion and degradation rate) of a morphogen in SynBioTIC is directly part of the behavior of the corresponding morphogenetic field. It must therefore be specified as part of an application program. Each application program therefore has the potential to require finding or engineering its own set of morphogens with specific properties. This approach also impacts scalability: rescaling a program requires new

morphogens with appropriately rescaled properties. By contrast, in my approach, the behavior of a morphogenetic field is an abstraction atop its corresponding physical morphogen. This layer of abstraction allows for a common library of morphogens that can be reused across programs and scales. This advantage is one of my central contributions and is discussed further in section 3.1.4.

Disset et al. [19] also explore artificial morphogenesis involving literal morphogens, again in simulation. As in the SynBioTIC project and my own work, they model Fickian diffusion and uniform degradation. The emphasis of their work is different from mine, however. They use evolutionary algorithms to explore the usefulness of a diversity of differently-specialized cells in a growing organism. Their simulated cells are controlled by idealized GRNs, and have the ability to metabolize, share resources, divide and apoptose, but not to move. Their approach to cell differentiation, like mine, requires extra morphogens so that communication can refer separately to different differentiation states.

2.5.4 Molecular Communication

Compared to the above examples of morphogen-like communication, molecular communication (MC) is a more general term. Molecular communication takes inspiration from communication systems in biological cells, both within and between organisms. As in biological cells, information is carried by molecules secreted by one agent and detected by another. Molecules may be transported actively (such as to particular neighbors or through a circulatory system) or passively, by advection and/or diffusion [30]. Morphogen communication is therefore a type of MC. Here I consider other types of MC and compare various MC information encoding schemes with my approach. Even these less morphogen-like MC systems are relevant, because they often assume similar communication environments and agent capabilities as in my work.

MC may be used in the contexts of different kinds of multi-agent systems. The MC literature most relevant to my work relates to amorphous computing. Wiedermann and Petr  [162] in particular suggest a model, in the amorphous computing context, that shares some communication-related assumptions with my own. They envision chemical signals diffusing in an aqueous environment, available to neighbors within a certain radius. Communication is strictly one-way: agents cannot discern the source of these chemicals and have no way to acknowledge receipt of information. Communication is asynchronous; agents may have similar timing mechanisms, but cannot assume they are synchronized. Finally, chemicals eventually degrade or decay into other species.

MC models encode information in different ways [30]. The amount of information conveyed by the chemical identity of a molecule varies. At one extreme, entire messages can be encoded in complex molecules such as DNA. At the other extreme, a single chemical may be used so that all information is encoded in changes in that species’ concentration over time. Between these extremes are schemes that, as in my approach, use a relatively small number of distinct chemical species.

Most MC schemes are crucially different from mine in that they convey serialized digital information. MC schemes typically divide time into communication intervals; within each interval, a symbol may be encoded by the concentration, identity, or timed release of molecules [30]. Messages then consist, as in digital communication, of sequences of symbols conveyed over time. Intersymbol interference is therefore a challenge in such MC schemes [30]. By contrast, my computational model is analog²; different chemical species represent parallel information channels, with concentration representing continuous values varying continuously over time. The analog nature of my model is one way of circumventing the problem of intersymbol interference.

Wiedermann [161] describes another method of encoding information in MC based on quorum sensing. In this case, each agent waits for some threshold number of molecules to be sensed locally before entering a new state. It is this quorum sensing variant of MC that is most similar to my work. In both quorum sensing and my work, molecular identity identifies a communications channel rather than a symbol. In addition, the quantity of molecules secreted by each agent, rather than merely being part of a symbol encoding, has the potential to represent information in its own right. The prototypical example of such information is the density of neighboring agents.

One difference between MC quorum sensing and my method is that, with quorum sensing, quantity of molecules is typically given a binary interpretation depending on whether or not a threshold is exceeded. In contrast, morphogen levels in my model are given a spatially continuous interpretation as field function values at each point in space. Another difference is that Wiedermann [161] are assuming that the molecules are well-mixed in the system. By contrast, my model depends crucially on the inhomogeneity of secreted morphogens in the domain. A less important difference is that Wiedermann assumes agents to be so small that it is important to model individual molecular binding events. In contrast, I assume agents are large enough that it is meaningful to speak of the local concentration of morphogens. However, I do model individual bindings in my simulator’s

²By *analog*, I mean those models of computation in which at least one domain is continuous. Different fields in the Morphogen language have distinct identities, so the language is discrete in that regard, but it is fully continuous in time, space, and field values. The computational model targeted by my compilation approach, discussed in chapter 3, is additionally discrete in tissue mass, which is represented by discrete agents; otherwise, it is a similarly continuous model.

sensor model (section 4.6). Other models for diffusive MC channels are reviewed by Jamali et al. [48].

Other quorum sensing models exist that are not explicitly part of the MC literature but have in common with the above studies an interest in distributed computation. Bernardini et al. [10] introduce quorum sensing into a P systems (membrane computing) model. Like Wiedermann [161], their primary interest is in establishing Turing-complete computation in a non-spatially-explicit environment. Therefore, despite the common interest in MC, their goals are similarly distant from my interest in embodied and spatially explicit computation. Gómez et al. [38] propose use of MC in the context of multi-agent spatial navigation tasks, which would seem to bring MC closer to my goals. However, their focus on externally designed gradient fields and lack of explicit agent control laws distinguishes their aims from mine. In addition, quorum sensing is not the basis for navigation in their work, but rather an improvement on more traditional navigation based on environmental cues.

Michelusi [87] consider a model of quorum sensing based on autoinducers in bacterial communities. They conclude that quorum sensing would be suitable for future collections of nanomachines to sense population density. Their model is not spatially explicit and so does not address local estimation of a heterogeneous density field. My model shares the idea of density estimation through the secretion and sensing of a molecular signal that is degraded in the environment. But in contrast to Michelusi’s model, my model describes density estimation using chemicals with known spatial profiles, based on the diffusion and degradation rates of the molecules used.

2.5.5 Electromagnetic Signal Strength

Besides artificial morphogens, I propose radiation as an alternative physical embodiment of SPH smoothing functions. Van der Klauw [155] measure radio received signal strength (RSS) as part of their method for a robot in a swarm to localize its neighbors relative to itself. Their use of RSS is inspired by odors (such as pheromones) in nature, which is similar to my inspiration in biological morphogens. Their method is crucially different from mine in that they use RSS to *localize* agents, whereas I propose using radiation to *avoid* the need to localize agents. Relatedly, their method involves an agent measuring RSS separately with each neighbor, whereas my method proposes using summed radiation intensity over all neighbors.

Mijalkov et al. [88] explore how different delays in agents’ response to light affects collective swarm behavior. Each agent emits visible light in all directions and simultaneously measures the

summed intensity of light from its neighbors. Agents respond to light levels by altering the frequency of random turns, creating a run-and-tumble dynamic inspired by chemotactic bacteria. Although this dynamic is not closely related to my work, the mechanism shares key similarities with my proposed method. Specifically, each agent senses the naturally summed light intensity produced by all its neighbors, and needs no concept of having individual neighbors. The authors successfully test their method with real robots, which supports the viability of my idea, at least for visible light. Also, the authors note that their fields of light intensity could be replaced by radially decaying fields of sound intensity or chemical concentration, which is consistent with my idea that natural smoothing functions could be realized with different types of physical fields.

Chapter 3

Theory

Chapter 1 concluded with a summary of my two primary research contributions:

- a compilation technique for PDE-based artificial morphogenetic programs, based on a generalization of SPH robotics, and
- an embodied communication protocol to simplify communication in SPH robotics and improve its scalability.

The present chapter details and elaborates on these research contributions. I begin in section 3.1 with the second of the two contributions, introducing an embodied communication scheme based on what I term *natural smoothing functions* or NSF. Sections 3.2 and 3.3 introduce two versions of NSF based on radiation and diffusing chemicals, termed *radiative smoothing functions* (RSFs) and *morphogen smoothing functions* (MSFs), respectively. SPH control based on NSF in general, and on MSFs in particular, introduces challenges. Two of these challenges I address in section 3.4. After this latter section, I return in section 3.5 to my other primary contribution, the generalization of SPH robotics as a compilation technique for Morphgen. The five sections after that one (sections 3.6 to 3.10) introduce extensions to the Morphgen language, both to expand descriptive power and to address other challenges involved with NSF (especially MSFs). The material presented in these five sections, in addition to that in section 3.4, represent secondary research contributions in support of the primary contributions repeated above. With section 3.11, the chapter concludes by drawing together these various corrections and language extensions to more precisely describe the compilation process.

3.1 Natural Smoothing Functions

If an agent can produce a physical field around itself having certain properties, then implicit in this field is a natural smoothing function, or NSF. I will refer to such physical fields as *NSF fields* to distinguish them from the potentially more abstract fields which NSF fields may be used to implement, as described in the next section. The NSF field around a given agent must satisfy the following:

- The strength or intensity (I will say *value*) of the NSF field must decrease smoothly and monotonically with distance from the agent.
- The NSF field’s value must approach zero with distance, and quickly enough that the integral of value over all space converges.
- The NSF field must be roughly symmetrical around the agent.
- The agent must have control over the amplitude of the NSF field independently of its shape.

Finally, the NSF fields of different agents must sum naturally in the environment, and each agent must be able to sense locally both the value and gradient of the summed NSF field. If these requirements are met, then groups of such agents can implement SPH robotic control. The first three of these requirements are derived from general requirements for SPH smoothing functions; see Liu and Liu [67] for a discussion. Another key requirement is that the integral of the smoothing function be unity, which I address in the next section.

In later sections, I explore one possibility in depth: that of agents that release and detect dilute substances that diffuse and degrade in a surrounding aqueous medium. This scenario is inspired by the cells of a developing embryo as it undergoes morphogenesis. I refer to the smoothing functions implicit in this mechanism as *morphogen smoothing functions* (MSFs), and to the resulting NSF-SPH approach as MSF-SPH. A second possibility for NSFs is for agents to produce and sense radiative fields (e.g., electromagnetic (EM) or acoustic) distinguished by frequency. Yet another possibility involves evanescent EM fields. I briefly elaborate on these latter two possibilities in section 3.2, but my work focuses on MSFs. Before introducing the latter in more detail in section 3.3, I discuss the possibilities of NSFs generally.

When speaking of NSFs in general, I will sometimes refer to an NSF *channel*, in the sense of a communication channel. An NSF channel is a concrete instance of an NSF field that agents can

distinguish from others. With respect to morphogens, each separately sensed chemical substance is a channel; with respect to radiative fields, each separate frequency band is a channel.

3.1.1 SPH Estimates

In contrast to the term *NSF field*, I use the term *field* by itself (or sometimes *abstract field* to emphasize the distinction) to describe more abstract entities.¹ These are the fields which traditional SPH seeks to simulate and which SPH swarm robotic control seeks to generate. In the context of artificial morphogenesis, these fields define tissues. Examples in this context are mass density, velocity, and the entities that play the role of ordinary variables in a Morphgen program. (Some of these examples, such as the density and velocity fields, may seem at first to be concrete and not abstract, referring to physical entities. But the term field refers to notional continua of these quantities. For example, the density field is considered to have a value at every point in the domain, whereas in reality we have only a finite collection of discrete agents.)

A diversity of SPH-based estimates of fields and their derivatives exist. In general, an SPH estimate at a point is the sum over neighboring particles of a smoothing function, or one of its derivatives, multiplied by some expression involving various spatially-varying quantities. Each of these quantities (e.g., mass, density, or velocity) is associated either with the location of the desired estimate or with the location of a given neighboring particle. Some SPH estimates require only quantities associated with the locations of neighboring particles, and such forms permit ready incorporation into the NSF framework. The most important example is the estimate of a field itself at location $\mathbf{x}$:

$$\langle f(\mathbf{x}) \rangle = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j), \quad (3.1)$$

where m_j , ρ_j , and $f(\mathbf{x}_j)$ are the mass of, local density at, and field value at, each neighboring particle j , and r_j is $\|\mathbf{x} - \mathbf{x}_j\|$.

For nonnegative-valued scalar fields, we can perform this estimate using the NSFs described in section 3.1. If an agent j controls the amplitude of its NSF field so that it has unity integral, then the value at $\mathbf{x}$ of the NSF field produced by agent j is equivalent to the evaluation of $W(r_j)$, where $W(r)$ is a smoothing function whose shape is determined by the physical nature of the NSF field. If instead agent j controls the amplitude of its NSF field so that its integral is $\frac{m_j}{\rho_j} f(\mathbf{x}_j)$, then the value at $\mathbf{x}$ of the NSF field produced by agent j is the entire term $\frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j)$. Because the NSF

¹Both of these uses of *field* are in the sense of quantities associated with every point in a spatial domain, as in physics. Nowhere in this work do I refer to fields in the sense of an algebraic structure.

fields of each agent j sum naturally in the environment, the total NSF field value at $\mathbf{x}$ is the desired local estimate of an abstract field, $\langle f(\mathbf{x}) \rangle$.

For agent j to control the amplitude of its NSF field as above, it must be able to estimate local density ρ_j . Traditional SPH offers multiple methods for estimating density, the simplest of which is known as “summation density” and can be adapted to the NSF framework with the use of an additional NSF channel. For summation density, $\rho(\mathbf{x})$ is simply substituted for $f(\mathbf{x})$ in eq. (3.1), leaving $\langle \rho(\mathbf{x}) \rangle = \sum_j m_j W(r_j)$. Thus if each agent j controls the amplitude of an NSF channel so that its integral is m_j , then the naturally summed value of this NSF field at a point $\mathbf{x}$ is the desired $\langle \rho(\mathbf{x}) \rangle$. Each agent makes this estimate by measuring the local value of the corresponding NSF channel at its own location, and substitutes this estimate for ρ_j in determining the amplitude with which it should produce each NSF channel corresponding to some abstract field $f(\mathbf{x})$.

3.1.2 Spatial Derivatives

Because eq. (3.1) is valid at each point in the domain, a spatial derivative of the field of SPH estimates is an estimate of that derivative in the corresponding field function. For example,

$$\langle \nabla f(\mathbf{x}) \rangle = \nabla \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j). \quad (3.2)$$

In traditional SPH, where a separate calculation is needed for an SPH estimate at each point, the above fact is not directly useful. Instead, much of the value of traditional SPH comes from the ability to rearrange expressions so that derivatives are applied only to the smoothing function and can be taken analytically. For example,

$$\langle \nabla f(\mathbf{x}) \rangle = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) \nabla W(r_j). \quad (3.3)$$

By contrast, in the NSF framework, SPH estimates of abstract fields are physically embodied by the total NSF field at a given point. Therefore, if technologically feasible, an agent can make an SPH estimate of a spatial derivative by sensing that derivative in the local NSF field. In particular, I assume agents can sense NSF field gradients locally, for example using differences between multiple sensors at their boundary, such that eq. (3.2) is directly useful.

Where such direct estimates are not practical, it may be possible to manipulate other methods from the SPH literature into forms compatible with the NSF framework. My adaptation of a

Laplacian estimate to work with NSF-SPH is the most important example. Huang et al. [46] derive an SPH form of the Laplacian operator requiring no derivatives. In two dimensions:

$$\langle \nabla^2 f(\mathbf{x}_i) \rangle = \frac{2}{\alpha} \sum_j (f(\mathbf{x}_j) - f(\mathbf{x}_i)) W(r_{ij}) \frac{m_j}{\rho_j},$$

where α is a constant that depends on an integral relating to a given smoothing function. Although it is not possible in the NSF framework to take the pairwise difference as shown, we can rearrange and note that $\sum_j \frac{m_j}{\rho_j} W(r_{ij})$ is just an SPH estimate of unity, and so can be replaced by 1. This gives us a form we can calculate in the NSF framework. For agent i ,

$$\begin{aligned} \langle \nabla^2 f(\mathbf{x}_i) \rangle &= \frac{2}{\alpha} \left(\sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_{ij}) - \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_i) W(r_{ij}) \right) \\ &= \frac{2}{\alpha} \left(\sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_{ij}) - 1 \cdot f(\mathbf{x}_i) \right) \\ &= \frac{2}{\alpha} (\langle f(\mathbf{x}_i) \rangle - f(\mathbf{x}_i)). \end{aligned} \tag{3.4}$$

(For our two-dimensional MSF described in section 3.3, following the derivation in Huang et al. [46] gives $\alpha = \pi \int_0^\infty W(r, h) r^3 dr = 2h^2$.)

This Laplacian estimate only makes sense for fields other than density. In the case of density, an agent has no knowledge of the field other than its SPH estimate, so there is no $f(\mathbf{x}_i)$ available for use in eq. (3.4), only $\langle f(\mathbf{x}_i) \rangle$. I do not expect this limitation to be severe in practice. Laplacian estimates are crucial in the context of elliptic PDEs; for example, if we want A , a field other than density, to obey the heat equation, we write

$$\frac{\partial A}{\partial t} = k \nabla^2 A,$$

where k is a constant. But because of the Lagrangian nature of SPH, if we want the density field to obey the heat equation, we need not write $\partial \rho / \partial t = k \nabla^2 \rho$. (Indeed we cannot write it, unless agents can be created or destroyed, which I explore in section 3.7.) Instead, in a Lagrangian system, the natural way to describe a density field that obeys the heat equation is

$$\mathbf{u} = -k \frac{\nabla \rho}{\rho},$$

where $\mathbf{u}$ is the velocity field, with no Laplacian estimate needed. That said, future work may explore the possibility of constructing a Laplacian estimate, similar in spirit to eq. (3.4), through the use of two density estimates. The two density estimates would be made using distinct NSF’s, one with a longer smoothing length and one with a shorter.

3.1.3 Signed and Vector Fields

Because NSF fields can have only nonnegative value, the natural interpretation of such a value is as a representation of a nonnegative scalar. But in traditional SPH, both signed and vector abstract fields are used routinely, such as for velocity. Such fields can be represented in NSF-SPH, though at some cost.

Signed scalar fields can be represented by either of two approaches. A finite range of negative scalars may be represented by introducing an offset, interpreting some arbitrary positive NSF value as representing zero. In addition to forcing a hard limit on the range of negative numbers that can be represented, this approach may be inefficient, assuming that agents expend more resources in producing larger field values.

Alternatively, by using two NSF channels, we can naturally represent fields with negative values. One channel would represent the magnitude of positive values, the other channel the magnitude of negative values. A given agent’s own value for this abstract field cannot be both positive and negative at once, so it will produce an NSF field in either one NSF channel or the other at a given time. By linearity of differentiation, all SPH derivative estimates are made by subtracting the derivative of the negative-representing NSF field from the derivative of the positive-representing NSF field. For example, say that positive values of abstract field X are represented by concentrations of morphogen A and negative values by the concentration of morphogen B . Each of these concentration fields is an NSF field; to foreshadow the notation in section 3.3, I denote them ϕ_A and ϕ_B . Then an agent’s NSF-SPH estimate of the gradient of X , that is, $\langle \nabla X \rangle$, is just $\nabla \phi_A - \nabla \phi_B$.

In addition to signed scalar fields, we can also represent vector fields with additional channels and some shared directional information. In three dimensions, we can represent each Cartesian coordinate with a channel or pair of channels, depending on whether we are using an offset approach or the above approach with separate positive and negative channels. Thus in two dimensions, a vector field would require two or four channels, and in three dimensions we would need three or six. (Cartesian coordinates are used because, for our SPH summations, we rely on the natural

summation of each channel in the environment. Among orthogonal coordinate systems, only Cartesian coordinates allow the summing of vectors by the summing of their components.)

Unfortunately, even such a multitude of channels is not enough to represent vector quantities; we also need directional information that is shared among agents. If agents were to produce component NSF fields with respect to their individual headings, the sums of these NSF fields would be meaningless. (It is a strength of the NSF-SPH approach that, *unless* we need to represent vector quantities, agents have no need for such shared directional information.) A directional cue might be gravity, the Earth’s magnetic field, or a light source, accompanied by appropriate sensors. To establish a shared orientation, one such cue is needed when working in two dimensions, and two such cues for three dimensions. So for example, agents working in three dimensions in a laboratory environment might take one cue from gravity and the other from a light shone perpendicular to gravity.

If the requirements are met to represent vector fields, it is straightforward to construct estimates of various vector derivatives from scalar derivatives of their components. An agent can construct a Jacobian matrix by projecting its estimates of the gradients of the component scalar fields onto the coordinates determined by the shared directional cues. Divergence and curl are simple functions of this Jacobian matrix. The vector Laplacian is even simpler; it is just the sum of the Laplacian estimates for each component.

The possibilities of working with signed, and especially vector, fields are worthy of further research. But for the present work, I focus on nonnegative scalar fields. Among the contributions represented by my work is a partial demonstration of the expressive possibilities of Morphgen, even when restricted to nonnegative scalar fields.

3.1.4 Benefits of NSF Framework

NSF-based SPH robotic control addresses communication challenges and inefficiencies in traditional SPH robotic control. Traditional methods require each agent, for each SPH estimate it makes, to communicate separately with $\mathcal{O}(n)$ other agents, where n is the number of agents in its neighborhood. In contrast, NSF-based agents need only sense an NSF field value locally, an $\mathcal{O}(1)$ operation. In establishing a separate data link with each neighbor, a traditional agent must avoid communication interference with its other neighbors, all of which may be simultaneously trying to establish links with *their* neighbors. By contrast, NSF-based agents communicate implicitly by

sensing an NSF field locally and manipulating their contribution to it; therefore, they do not require any explicit data links or even a concept of having individual neighbors.

We can view these advantages over traditional SPH control as the benefits of embracing an embodied approach to artificial morphogenesis. In the agent-to-agent communication of traditional SPH control, information is an abstraction atop the physical fields chosen as communication media. Overcoming communication interference is thus a struggle against the way these fields interact in space. By contrast, the NSF framework works with, rather than against, the interaction (as well as diminution with distance) of the physical NSF fields available to a system. For example, morphogen-based SPH control takes advantage, in two key ways, of the behavior of morphogens in the aqueous environments typical of a growing embryo. First, it offloads pairwise distance estimates to the natural diminution with distance of a dilute substance as it diffuses and degrades. Second, it offloads a summation loop across many neighbors to the natural summation of dilute substances from multiple sources.

3.2 Radiative and Evanescent Smoothing Functions

Although I explore only MSFs in detail, alternative NSF implementations are possible in principle. I discuss two possibilities here. The first involves radiation with energy dissipation, for example, EM or acoustic. I call the NSFs implied by radiation *radiative smoothing functions* (RSFs), and I refer to the resulting approach to NSF-SPH as RSF-SPH. I also briefly mention the possibility of NSFs implied by evanescent EM fields.

In the context of radiation, the physical quantity that can imply an NSF is *scalar irradiance*. Scalar irradiance has dimensions power per unit area, and characterizes the total amount of radiant flux, or power, impinging on a point. Loosely speaking, it is the sum of irradiance (also known as intensity) coming from all directions. More precisely, scalar irradiance is the integral of radiance over all directions, radiance being radiant flux per unit area per solid angle. In the EM case, scalar irradiance is not the same as what is known in physics as the EM field (it is, rather, related to the square of the EM field). However, in the broader sense of an assignment of a value to each location in a spatial domain, scalar irradiance is a field. Furthermore, I conceive of NSF-SPH in general in terms of abstract fields implied by physical fields. For this reason, when I refer to the *physical field* in the context of RSF-SPH, I mean scalar irradiance.

It is natural to represent different NSF channels as different frequency bands. The derivative of scalar irradiance with respect to frequency is scalar spectral irradiance. Properly speaking then, the physical quantity that will represent a given RSF channel is scalar spectral irradiance integrated over some frequency band. Its dimensions are still power per unit area. For simplicity, I will leave “integrated over some frequency band” implicit for the rest of this section.

I also consider the possibility of waves that propagate in only two dimensions, such as surface waves. In this case, I use *scalar irradiance* to mean the two-dimensional analogue of the above quantity, with dimensions power per unit length.

Assuming waves are radiating symmetrically in three dimensions, scalar irradiance varies with distance from source according to the inverse square law. Unfortunately, this variation is not enough on its own to imply a valid smoothing function, because the integral of an inverse-square function over two- or three-dimensional space is divergent. (The integral of scalar irradiance over one dimension is convergent, but SPH control is not likely to be useful in a single dimension.) If waves are propagating in two dimensions, scalar irradiance falls as the inverse of distance instead of the inverse square. Both the one- and two-dimensional integrals of scalar irradiance are divergent in this case.

These divergent integrals preclude the use of idealized radiation as NSFs. However, except for the case of EM radiation in a vacuum, some of a propagating wave’s energy will be dissipated in its medium. If this dissipation is uniform, it introduces an exponential attenuation to the variation of scalar irradiance with distance (according, in the example of EM radiation, to the Beer-Lambert law). This attenuation causes scalar irradiance to fall fast enough with distance that we may construct NSFs. In a laboratory environment, one may imagine controlling the shape of EM RSFs in particular by filling the domain with a solution or well-mixed suspension, in a gas or liquid medium, of a suitably energy-absorbing substance.

Integrability is not the only requirement for an NSF; the physical field must also sum naturally in the environment. For scalar irradiance, this requirement holds with two caveats. First, the radiation from different sources must be incoherent. I expect incoherence to be the usual situation when agents are not radiating at any particular phase and are not located at any particular fixed distances from each other. Second, agents’ measurements of incident radiance must be isotropic in order to measure scalar irradiance.

An agent may measure scalar irradiance in different ways depending on implementation details. For example, an agent may measure scalar irradiance at radio frequencies using orthogonally

arranged omnidirectional antennas. In the case of visible light, radiometers often measure scalar irradiance with a diffusing sphere mounted on a thin stick. With multiple such spheres, derivatives in spectral irradiance could be computed by simply subtracting readings and dividing by separation distance. However, at smaller scales and considering collisions, sensors in a few hemispherical diffusers around an agent’s surface may be more practical than spheres on stalks. Calculating a gradient estimate from such sensor readings would be more involved than a simple subtraction and division, and I leave the details for future work. I speculate that such estimates could be more precise because of the anisotropic nature of the individual sensor readings.

In two dimensions, scalar irradiance at distance r from a point source, in terms of source power P and assuming uniform dissipation, is

$$I(r) = \frac{P \exp(-r/h)}{2\pi r}, \quad (3.5)$$

where the form of the constant h is chosen so that it plays the role of a smoothing length in an SPH smoothing function. For three dimensions, the corresponding form for scalar irradiance is

$$I(r) = \frac{P \exp(-r/h)}{4\pi r^2}. \quad (3.6)$$

The RSFs implied by these physical fields will be proportionate to them. Furthermore, they must integrate to one, so we can find them by normalizing eqs. (3.5) and (3.6). Dividing our expressions for scalar irradiance by their integrals over the spatial domain (in two or three dimensions as appropriate) gives the first and third rows of table 3.1.

Next we need to derive the source power that agents should use such that RSFs are implicit in the physical fields they generate. We want the SPH estimate of a field variable at a point to be proportional to the natural summation at that point of contributions to scalar irradiance from all agents:

$$\langle f(\mathbf{x}) \rangle = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j) = \frac{\sum_j I_j(\mathbf{x})}{\gamma}, \quad (3.7)$$

where $I_j(\mathbf{x})$ is the scalar irradiance measured at point $\mathbf{x}$ attributable to agent j and γ is a constant. The SPH estimate is an abstraction with no physical units, so γ has units of scalar irradiance. The value of γ is arbitrary, and can be set according to technical convenience. Next, we equate the

Table 3.1: Idealized radiation smoothing functions. Distance r_0 is the effective radius of an agent, or the minimum distance from a radiation source to any agent’s sensor. Γ is the incomplete gamma function.

Dimensionality	Smoothing function
Two dimensions, 2D radiation	$W(r) = \frac{\exp(-r/h)}{2\pi hr}$
Two dimensions, 3D radiation	$W(r) = \frac{\exp(-a/h)}{\pi a^2 (\exp(-r_0/h) + 2\Gamma(0, r_0/h))}$, where $a = \max(r, r_0)$
Three dimensions	$W(r) = \frac{\exp(-r/h)}{4\pi hr^2}$

summands and substitute for $W(r_j)$ and $I_j(\mathbf{x})$. In two dimensions, this gives

$$\frac{m_j \exp(-r/h)}{2\pi\rho_j h r} f(\mathbf{x}_j) = \frac{P \exp(-r/h)}{\gamma 2\pi r},$$

where, for readability, I use r in place of r_j to mean $\|\mathbf{x} - \mathbf{x}_j\|$. Solving for P gives the source power needed:

$$P = \gamma \frac{m_j f(\mathbf{x}_j)}{\rho_j h}. \quad (3.8)$$

In three dimensions, the corresponding substitutions for $W(r_j)$ and $I_j(\mathbf{x})$ give:

$$\frac{m_j \exp(-r/h)}{4\pi\rho_j h r^2} f(\mathbf{x}_j) = \frac{P \exp(-r/h)}{\gamma 4\pi r^2}.$$

The different factors in the denominators cancel, so the same source power given in eq. (3.8) for two dimensions also applies in three dimensions.

We may be interested in a situation in which radiation occurs in three dimensions but agents lie in a plane, especially if agents are moving along a surface. In this situation, it is tempting to simply use the three-dimensional RSF, reasoning that agents within a plane are, trivially, also within three-dimensional space. However, this idea is in error, because SPH assumes that agents are spread throughout all dimensions of the smoothing function. One way to see this is to consider a plane filled with evenly distributed agents at some average density. If we make density estimates by counting agents within discs of varying size, clearly the expected density estimate should be the same regardless of disc size. But if we embed this plane in a three-dimensional space, the expected number of agents found per spherical volume of space would vary with the radius of the sphere, because area scales with r^2 but volume scales with r^3 . Therefore, using a three-dimensional smoothing function to estimate our fixed two-dimensional mass density would give varying results depending on smoothing length.

When radiation occurs in three dimensions, we can still construct a valid two-dimensional smoothing function by considering just the slice of the radiation field that intersects the tissue plane. However, the two-dimensional integral of a cross-section of three-dimensional scalar irradiance is divergent—not because the field falls off too slowly with distance, but because the singularity at the source is not integrable. We can work around this problem if we assume that no scalar irradiance measurement will ever be taken more than some fixed distance r_0 from a radiation point source, including an agent’s measurement of its own radiation sources. For example, perhaps our agents are

puck-shaped robots with lights mounted on the middle of their upper surface and sensors around their circumference. (Of course, no real radiation source is truly a point source anyway, but the larger a distance r_0 we can establish, the more accurate will be our model of the source as a point.)

If r_0 is the minimum measurement distance from a point source, we can treat the RSF as flat below this radius; that is, $W(0) = W(r) = W(r_0)$ for $0 < r < r_0$. This interpretation removes the offending singularity and allows us to normalize the RSF so that it has unity integral. The resulting RSF is shown in the middle row of table 3.1.

Because no measurements of the RSF will be taken at radii less than r_0 , flatness does not damage accuracy as it otherwise would. In fact, this flatness is beneficial, and we should make r_0 no smaller than necessary. Once the integral is normalized, a smaller r_0 results in a higher maximum RSF value. But if an agent measures its own RSF contribution at a distance larger than r_0 (call this distance b), then it will sense a scalar irradiance lower than $W(r_0)$; that is, $W(b) < W(r_0)$. Furthermore, we treat agents as points in our NSF derivations, so an agent's SPH estimates are considered to be for location $r = 0$, meaning its own NSF contribution to its measurements should be $W(0)$. Therefore, if $W(0) = W(r_0) > W(b)$, then we are effectively assuming that the agent will receive a higher contribution from its own NSF field than is possible.

To find the correct source power in this hybrid dimensional case, we follow the same steps as before. In this case we are only interested in the part of the smoothing function that is proportional to physical scalar irradiance, so we ignore the flat inner disc in the expression in the middle row of table 3.1. Substituting this expression into eq. (3.7) for $W(r_j)$, substituting the three-dimensional scalar irradiance from eq. (3.6) for $I_j(\mathbf{x})$, and equating summands as before, we have:

$$\frac{m_j \exp(-r/h)}{\pi \rho_j r^2 (\exp(-r_0/h) + 2\Gamma(0, r_0/h))} f(\mathbf{x}_j) = \frac{P \exp(-r/h)}{\gamma 4\pi r^2}.$$

Solving for P gives the source power required:

$$P = \gamma \frac{4m_j f(\mathbf{x}_j)}{\rho_j (\exp(-r_0/h) + 2\Gamma(0, r_0/h))}.$$

The derivations of the three RSFs in table 3.1 rely on other simplifying assumptions besides those I have mentioned. For example, they ignore near-field effects and scattering by the medium. Accounting for such complications would require deriving new RSFs and corresponding source power. However, having done that, the resulting RSFs would still be valid for use in NSF-SPH. With respect

to radiation, the key assumptions for valid RSFs are uniformity of medium, isotropic radiation and measurement, and summation of radiation from different sources.

Besides radiative fields, another electromagnetic possibility for NSFs involves evanescent fields. These fields are formed when EM waves undergo total internal reflection at an interface due to high angles of incidence [42]. Although the waves do not travel beyond the interface, standing waves do create an EM field, called evanescent, on the other side of the interface. The strength of this field drops exponentially with distance from the interface.

The generation, measurement, and nature of evanescent fields are specialized topics, and I do not attempt to evaluate their practicality in NSF-SPH use. I offer instead a few speculative observations. Suppose an agent could generate EM waves within its body such that they reflected internally, generating an evanescent field beyond. This evanescent field would be a valid NSF with two useful characteristics. First, the exponential drop in field strength with distance would obviate the need for additional attenuation in the medium in order for the field to have a convergent integral. Second, because evanescent waves do not propagate, they do not radiate energy, potentially making evanescent NSFs more efficient than RSFs.

3.3 Morphogen Smoothing Functions

3.3.1 Idealized MSFs

A third possibility for NSFs, and the one that I focus on for the remainder of this work, is the use of diffusing chemical substances. Because of the morphogenetic inspiration for this work, I call these substances *morphogens*, even though a morphogen is properly a substance used for signalling in biological morphogenesis. I refer to the NSFs based on these morphogens as MSFs, as mentioned in section 3.1.

Throughout this section, several simplifying assumptions are made:

- Each agent is a point.
- Each morphogen obeys Fick’s law for homogeneous, isotropic diffusion (i.e., change in concentration is a constant proportion of the Laplacian).
- Degradation of each morphogen is uniform in the medium and proportional to morphogen concentration.
- Each agent’s speed is negligible relative to morphogen diffusion and degradation rates.

- Morphogen production, diffusion, and degradation are in equilibrium.

Later, I introduce corrections for some violations of the latter two assumptions.

At what rate should an agent produce morphogen so that the resulting concentration field implies a valid NSF? We want the SPH estimate of a field variable at a point to be proportional to the summed concentration of an associated morphogen at that point:

$$\langle f(\mathbf{x}) \rangle = \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j) = \frac{\sum_j \phi_j(\mathbf{x})}{\gamma}, \quad (3.9)$$

where $\phi_j(\mathbf{x})$ is the morphogen concentration at $\mathbf{x}$ attributable to neighboring agent j and γ is a constant. As in section 3.2, the SPH estimate has no physical units, so γ has units of concentration. Once again, its value is arbitrary and can be set to satisfy technical convenience. Equating the summands, each agent must choose to produce morphogen at rate a_j such that

$$\frac{m_j}{\rho_j} f(\mathbf{x}_j) W(r_j) = \frac{\phi_j(\mathbf{x})}{\gamma}. \quad (3.10)$$

Let Φ_j be the total amount of morphogen in the environment due to agent j , at equilibrium:

$$\Phi_j = \int_{\Omega} \phi_j(\mathbf{x}) d\mathbf{x}.$$

If k is the morphogen's decay rate, then Φ_j is related to a_j by

$$\frac{d\Phi_j}{dt} = a_j - k\Phi_j \quad (3.11)$$

and therefore reaches equilibrium when

$$\Phi_j = \frac{a_j}{k}. \quad (3.12)$$

Solving eq. (3.10) for $W(r_j)$, integrating both sides, and considering the requirement of unity for the integral of $W(r_j)$, we have

$$\int_{\Omega} W(r_j) d\mathbf{x} = \int_{\Omega} \frac{\rho_j}{\gamma m_j f(\mathbf{x}_j)} \phi_j(\mathbf{x}) d\mathbf{x} = \frac{\rho_j}{\gamma m_j f(\mathbf{x}_j)} \Phi_j = \frac{a_j \rho_j}{\gamma k m_j f(\mathbf{x}_j)} = 1.$$

We can now solve for a_j to determine the correct production rate for agent j :

$$a_j = \gamma \frac{k m_j f(\mathbf{x}_j)}{\rho_j}. \quad (3.13)$$

Table 3.2: Idealized morphogen smoothing functions. The first row gives PDEs describing diffusion and degradation, and the second row gives general solutions of these PDEs. The last two rows give the smoothing functions themselves. The variable ϕ is morphogen concentration, and r is distance from source. J_0 and Y_0 are Bessel functions of the first and second kind respectively, of order zero. K_0 is a modified Bessel function of the second kind, of order zero. E is diffusion rate and k is decay rate, and in the final row the substitution $h = \sqrt{E/k}$ is made to obtain forms resembling the typical presentation of SPH smoothing functions.

	One dimension	Two dimensions	Three dimensions
Radial form of PDE	$E \frac{\partial^2 \phi}{\partial r^2} - k\phi = 0$	$\frac{E}{r} \frac{\partial \phi}{\partial r} + E \frac{\partial^2 \phi}{\partial r^2} - k\phi = 0$	$2 \frac{E}{r} \frac{\partial \phi}{\partial r} + E \frac{\partial^2 \phi}{\partial r^2} - k\phi = 0$
General solution	$\phi(r) = c_1 \exp(-r\sqrt{k/E})$ $+ c_2 \exp(r\sqrt{k/E})$	$\phi(r) = c_1 J_0(ir\sqrt{k/E})$ $+ c_2 Y_0(-ir\sqrt{k/E})$	$\phi(r) = c_1 \frac{\exp(-r\sqrt{k/E})}{r}$ $+ c_2 \frac{\exp(r\sqrt{k/E})}{r}$
Smoothing function	$W(r) = \frac{\exp(-r\sqrt{k/E})}{2\sqrt{E/k}}$	$W(r) = \frac{K_0(r\sqrt{k/E})}{2\pi E/k}$	$W(r) = \frac{\exp(-r\sqrt{k/E})}{4\pi r E/k}$
As above, with $h = \sqrt{\frac{E}{k}}$	$W(r) = \frac{\exp(-r/h)}{2h}$	$W(r) = \frac{K_0(r/h)}{2\pi h^2}$	$W(r) = \frac{\exp(-r/h)}{4\pi h^2 r}$

Under the simplified assumptions from the beginning of this section, we can solve a PDE describing a diffusing, degrading morphogen in order to learn the shape of the MSFs implied by our derived production rates. These PDEs and their general solutions are shown in the first two rows of table 3.2.

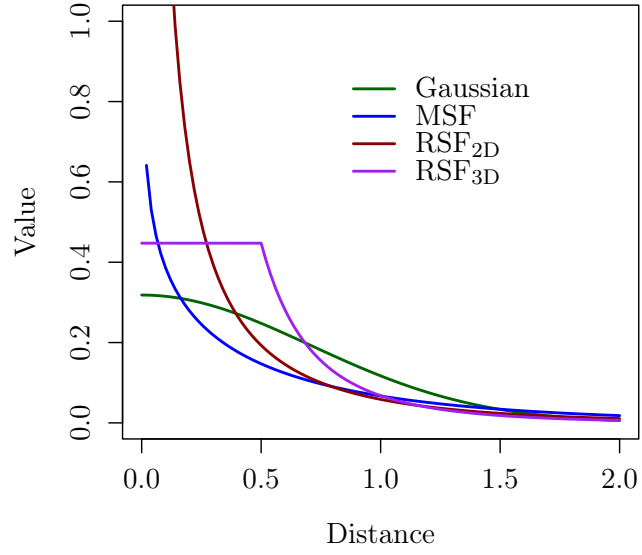
The growing parts of these general solutions are rejected as non-physical. Normalizing what remains gives the MSFs themselves, shown in the last two rows. In the second-to-last row, smoothing functions are given in terms of physical parameters: diffusivity E and degradation rate k . (I use E for diffusivity instead of the traditional D , to avoid confusion with the difference operator.) In the final row, E and k are replaced by a smoothing length $h = \sqrt{E/k}$. This choice allows h to play the same role as in traditional SPH smoothing functions, and it also matches a quantity known as λ in biological morphogenesis and variously called *signaling range* [94] or *decay length* [55].

The two- and three-dimensional versions of these smoothing functions are compared with corresponding Gaussian and radiative smoothing functions in fig. 3.1. Gaussian smoothing functions are considered archetypal in SPH literature. Although the smoothing length h is set to one for all kernels shown, this quantity has little practical meaning in comparing kernels with such different shapes as these.

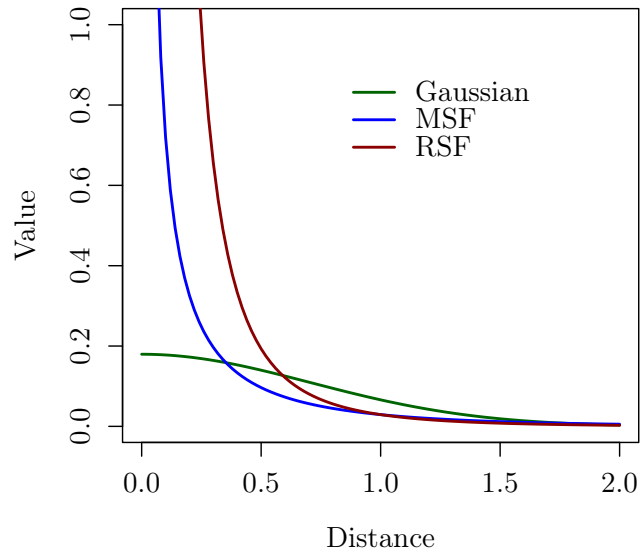
3.3.2 Practical Considerations

SPH is most accurate when each agent has an intermediate number of neighbors. Liu and Liu [67] cite 21 and 57 as ideal numbers in two and three dimensions, respectively. There is no straightforward way to translate such numbers to NSF-SPH, in which smoothing functions have much fatter tails than in traditional SPH. However, ensuring that morphogens' smoothing lengths are compatible with agent density is still important in NSF-SPH. Because morphogen smoothing length is determined by diffusivity and degradation rate, these two quantities will be central in engineering a practical MSF-SPH system.

In biological morphogenesis, effective diffusion rates are determined by multiple factors, including how restricted the fluid medium may be by closely-packed cells or by other compounds in the extracellular matrix that bind the morphogen [94]. However, for a given medium's temperature and viscosity, free diffusion rates are determined by particle size alone (for spherical particles, the Stokes-Einstein equation gives the precise relation). Free diffusivity in water at room temperature can be as high as $2000\text{--}3000\text{ }\mu\text{m}^2\text{ s}^{-1}$ for nitric oxide [166], perhaps the smallest natural morphogen. Decapentaplegic (Dpp) is one of the largest morphogens [58]; its diffusivity in water is thought



(a) Two dimensions



(b) Three dimensions

Figure 3.1: Comparison of idealized MSF, RSF, and Gaussian smoothing functions in two (a) and three (b) dimensions. $\text{RSF}_{2\text{D}}$ refers to a scenario in which radiation is constrained to two dimensions. $\text{RSF}_{3\text{D}}$ refers to a scenario in which agents operate in a plane but radiation propagates in three dimensions, as described in section 3.2.

to be similar (based on its size) to that of another protein with diffusivity $87 \mu\text{m}^2 \text{s}^{-1}$ [55]. This diffusivity is not much slower than $100 \mu\text{m}^2 \text{s}^{-1}$, which Müller and Schier [95] suggest as a typical aqueous diffusivity for protein morphogens. Non-protein morphogens tend to be smaller and have higher diffusivity; for example, cyclic adenosine monophosphate (cAMP), which acts as a morphogen in *Dictyostelium*, has diffusivity around $400 \mu\text{m}^2 \text{s}^{-1}$.

Degradation rates vary widely, and our knowledge of these rates is uncertain. Multiple mechanisms may degrade morphogens, and in addition to degradation, morphogen may also be cleared from the intercellular space by endocytosis. In addition, macroscopic observations often provide information about the relationship between effective diffusivity and degradation rate without determining both simultaneously. Reviewing kinetic studies of various protein morphogens, Kicheva et al. [54] give half-life estimates ranging from eight minutes to over four hours. (When degradation is uniform, its rate k and half-life $\tau_{1/2}$ are related as $k\tau_{1/2} = \ln 2$.) Some morphogens have half-lives as short as 10–15 s, including Dpp [94] and cAMP [108].

Unlike free diffusivity, which is determined by molecular size, degradation is governed by the chemical interaction between a morphogen and a catalyst, such as an enzyme, to which it binds. For this reason, I expect degradation rate, rather than diffusivity, to be the variable through which smoothing length will be engineered for use in MSF-SPH. A catalyst must convert a morphogen to a form that no longer binds to the agent’s sensors. In an ideal MSF-SPH system, this inactive form would be taken back up by the agent in some manner, such as endocytosis, where it would be converted back to its active form and re-secreted as needed.

The assumption that morphogen degradation is uniform is crucial for MSF-SPH. Without it, the concentration field would not be equal to the sum, over all agents, of the concentration profiles attributable to each agent individually. In that case, we could not have equated the summands in eq. (3.9) to obtain eq. (3.10). In chemical terms, this requirement is equivalent to a need for the interaction between morphogen and catalyst to follow first-order kinetics. To achieve first-order kinetics, the probability of a morphogen molecule encountering a free catalyst molecule must be similar for any morphogen concentration that will be encountered when running a given Morphgen program. One implication of this requirement is that engineering a desired degradation rate cannot be achieved by controlling catalyst concentration; instead, it must be achieved through the chemical identities of the morphogen and catalyst.

The requirement for first-order kinetics also plays a part in limiting the dynamic range of field values as represented by the morphogen concentration. Kinetics will not be first order if the ratio

of morphogen to catalyst concentration becomes so high that catalyst concentration is a significant factor in determining degradation rate. (This ratio will depend on how quickly a catalyst dissociates from its degraded ligand after an association event.) Therefore, catalyst concentration must be held above some minimum level relative to maximum expected morphogen concentration, and the feasibility of doing so will constrain dynamic range on the high end. On the low end, dynamic range will be constrained by the sensitivity of agents' sensors to low morphogen concentrations. (Other potential constraints on the high end are sensor saturation and the feasibility of high secretion rates.)

These various limitations on dynamic range will result in trade-offs in setting the value for γ in eqs. (3.9) and (3.13), which acts something like a gain control. If γ is too high, we risk violating upper-end constraints on dynamic range, and vice-versa. The inaccuracies resulting from such violations would be analogous to overflow and underflow errors. Fortunately, if these constraints are too severe, dynamic range can be extended by adding a second or third distinct set of receptors that bind the same morphogen but with different affinities; this solution is employed by biological cells [154].

Future work may address whether some useful system, in the spirit of MSF-SPH, may be able to accommodate second-order degradation kinetics. However, such a system might be too mathematically dissimilar to be considered a flavor of SPH.

3.4 Correcting for Transient Spurious Morphogen Smoothing Functions

Traditional SPH assumes no lag time in information flow, but two factors prevent morphogens from conveying information instantaneously among agents. The first is the time needed for production and degradation processes to reach steady state, which is a non-spatial consideration. The second is the time needed for diffusion to spread information spatially from an agent to its neighbors. These two factors lead to transient spurious MSFs, and I discuss each in turn. In neither case do I fully correct for transient effects; instead, I make simplifying assumptions that allow for the correction of the most detrimental aspect of each effect.

3.4.1 Changes in Production Rates

I first address the non-spatial transient effect of morphogen production being out of equilibrium with degradation, which occurs whenever production rates have recently changed (fig. 3.2). I make two simplifying assumptions to find a correction term for agents to add to their sensor readings to better approximate what those readings would be under equilibrium conditions. First, I neglect diffusion time; in other words, I assume that each agent instantaneously injects morphogen into its entire neighborhood according to the distribution of the idealized MSF. Second, I assume that the field variable values for all agents in a neighborhood are changing at the same rate a so that $f(\mathbf{x}_j, t) = at + b_j$ since time $t = 0$, and that until $t = 0$ the values were not changing and the system was in equilibrium.

(This latter assumption is justified by the smoothness of SPH fields. Just as traditional SPH models fields as smooth, SPH robotics tends to generate smooth fields. As we will see, the present derivation is the first of several examples in which I have found it helpful to exploit this smoothness by assuming that neighboring agents are similar.)

Based on these assumptions, we wish to find a correction term $g(t)$ such that $\langle f(\mathbf{x}, t) \rangle = \sum_j \phi_j(\mathbf{x}, t) + g(t)$. At a given time $t > 0$, from eq. (3.11) and eq. (3.13) the total mass of morphogen in the environment due to agent j obeys

$$\frac{d\Phi_j(t)}{dt} = r_j(t) - k\Phi_j(t) = \frac{km_j(at + b_j)}{\rho_j} - k\Phi_j(t).$$

This equation is solved by

$$\Phi_j(t) = ce^{-kt} + \frac{m_j}{\rho_j} \left(at + b_j - \frac{a}{k} \right). \quad (3.14)$$

Because the system was in equilibrium at $t = 0$, we can substitute from eq. (3.12) and eq. (3.13):

$$\Phi_j(0) = \frac{r_j(0)}{k} = \frac{m_j f(\mathbf{x}_j, 0)}{\rho_j} = \frac{m_j b_j}{\rho_j}.$$

After substituting into eq. (3.14) and solving for c , we have:

$$\Phi_j(t) = \frac{m_j a}{\rho_j k} e^{-kt} + \frac{m_j}{\rho_j} \left(at + b_j - \frac{a}{k} \right) = \frac{m_j a}{\rho_j k} e^{-kt} + \frac{m_j f(\mathbf{x}_j, t)}{\rho_j} - \frac{m_j a}{\rho_j k},$$

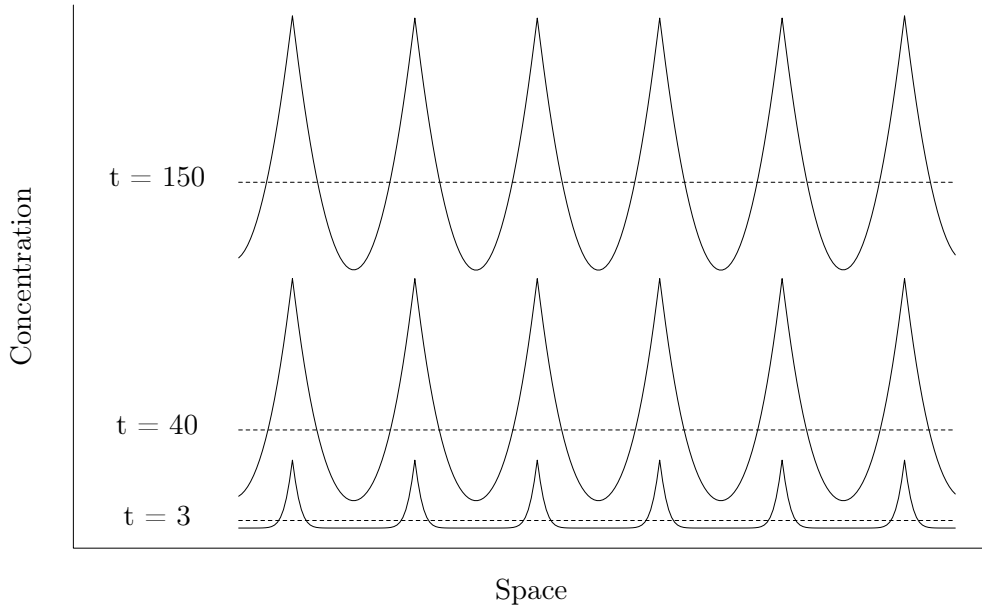


Figure 3.2: One-dimensional toy simulation to illustrate spatial vs. non-spatial aspects of non-steady-state conditions. Units are arbitrary. As time advances, agents secrete a morphogen at a constant rate from a uniform starting concentration of zero. A steady state, not shown, would eventually result. As total morphogen in the domain increases, the relative difference in concentration from agents' locations to points between agents decreases. The non-spatial correction factor derived here considers only total morphogen and ignores the spatial variation, equivalent to assuming that morphogen concentration is as shown by the dashed lines.

which we can solve for $f(\mathbf{x}_j, t)$:

$$f(\mathbf{x}_j, t) = \frac{\rho_j}{m_j} \Phi_j(t) + \frac{a}{k} (1 - e^{-kt}). \quad (3.15)$$

Substituting eq. (3.15) into eq. (3.1) and rearranging, a correct SPH estimate of $f(\mathbf{x}, t)$ under the non-equilibrium assumptions would therefore be

$$\langle f(\mathbf{x}, t) \rangle = \sum_j \Phi_j(t) W(r_j) + \sum_j \frac{m_j}{\rho_j} \frac{a}{k} (1 - e^{-kt}) W(r_j). \quad (3.16)$$

The left summation of eq. (3.16) is the SPH estimate of $f(\mathbf{x}, t)$ under the equilibrium assumption and is also $\sum_j \phi_j(\mathbf{x}, t)$, the sensed morphogen concentration at $\mathbf{x}$ (because we derived the production rate under the equilibrium assumption). The right summation of eq. (3.16) is just the SPH estimate of $\frac{a}{k} (1 - e^{-kt})$, but this quantity does not depend on j , so no SPH estimate is necessary for this term:

$$\langle f(\mathbf{x}, t) \rangle = \sum_j \Phi_j(t) W(r_j) + \frac{a}{k} (1 - e^{-kt}).$$

Thus we have the desired correction term:

$$g(t) = \frac{a}{k} (1 - e^{-kt}). \quad (3.17)$$

Each agent needs to keep track of $g(t)$, which is possible so long as it maintains an estimate of $\partial f(\mathbf{x}, t)/\partial t$. (In my simulation I simply have each agent remember one previous value of $f(\mathbf{x}, t)$ which it can divide by a known timestep.) To see this, we first take a time derivative: $dg(t)/dt = ae^{-kt}$. Note that we can rearrange eq. (3.17) to get $e^{-kt} = 1 - (k/a)g(t)$. We substitute this into the derivative along with $a = \partial f(\mathbf{x}, t)/\partial t$ to obtain

$$\frac{dg(t)}{dt} = \frac{\partial f(\mathbf{x}, t)}{\partial t} - kg(t), \quad (3.18)$$

which implies the needed update rule for agents to follow (in my simulation, $\Delta g(t) = \Delta f(\mathbf{x}, t) - kg(t)\Delta t$).

This correction term applies most clearly to estimating fields other than density, because secretion rates for density fields mostly do not change.² Furthermore, when an agent needs the

²There are a few occasions when secretion rates corresponding to density fields would be expected to change at a similar rate among agents in a neighborhood. One is at the beginning of a Morphogen program's execution, when all

value of a non-density field, it can simply use its stored value for that field. Therefore, I currently apply this correction term only to the Laplacian estimate described in section 3.1.2, because this is the only time when an agent needs to make an SPH estimate of a non-density field. Also, although more exploration is needed, allowing this correction term to change the sign of a Laplacian estimate appears to lead to instabilities in some applications. Therefore, when the correction would change the sign of the Laplacian estimate, the Laplacian is instead estimated as zero.

3.4.2 Agent Motion

The time it takes for morphogen diffusion to convey information spatially also leads to violations of steady-state assumptions whenever the position or production rate of an agent has recently changed. Here I address only transient effects due to agent motion, because in simulation it seems to be more detrimental than spatial effects due to production rate changes. As an agent moves, its contribution to the total morphogen field becomes stretched behind and compressed in front, with more complex distortions if the agent's path curves. Even more complex effects arise as a result of the particularities of an agent's body shape and instrument positions and the way a given fluid medium flows around an agent's body. Because such particularities imply implementation dependence of distortions, I do not pursue analytic corrections. Instead, I propose that agents be calibrated to correct for some of this distortion, an approach I test in simulation.

In informal testing, the most damaging aspect of these distorted MSFs is the effect an agent's own contribution to the morphogen field has on its SPH estimates. In addition, an agent has no knowledge of its neighbors' individual positions and velocities, limiting its ability to correct for distortions in neighbors' contributions to the morphogen field. For these reasons I focus calibration on correcting for distortions in an agent's own contributions to its SPH estimates.

The central idea of calibration is to have isolated agents move at various constant speeds, secreting a given morphogen at an arbitrary but constant rate. Once steady-state relative to the agent is reached, the agent then records the measured morphogen concentration and gradient, divided by the arbitrary secretion rate. A table of concentrations and gradients for different morphogens and speeds can be created in this way. Each entry of this table is a pair of concentration and gradient measurements. If M is a morphogen and s a speed, let $\text{CALIB}(M, s)$ refer to a lookup operation in this table. In general, s will not exactly match one of the speeds used in calibration;

agents are first brought online. The other is when tissue is in the process of differentiating (section 3.6). I have not yet tested the ideas expressed in this section in these scenarios, but they may be of benefit in the future.

in this case, a given implementation of this table lookup might return the entry for the nearest matching speed, or it might return an interpolated value. I denote the concentration and gradient parts of the returned pair as $\text{CALIB}(M, s)_{\text{VAL}}$ and $\text{CALIB}(M, s)_{\text{GRAD}}$, respectively.

During operation, such table lookups, multiplied by the agent's current secretion rate $\text{SECRATE}(M)$ for morphogen M , can be used to correct for an agent's distorted contribution to a morphogen field. If an agent's SPH estimate of the field value $f(x)$ corresponding to morphogen M is otherwise $\langle f(x) \rangle = \text{SENSE}(M)$, then its adjusted SPH estimate is

$$\langle f(x) \rangle = \text{SENSE}(M) + \text{SECRATE}(M) \times (\text{CALIB}(M, 0)_{\text{VAL}} - \text{CALIB}(M, s)_{\text{VAL}}).$$

This addition represents the additional concentration of an agent's own secreted morphogen that it would sense were it not moving. This adjustment is made whenever $\langle f(x) \rangle$ is needed in NSF-SPH. For example, it is used where $\langle f(x) \rangle$ appears in the NSF-SPH Laplacian estimate given in section 3.1.2. Similarly, if an agent's SPH estimate of the field gradient $\nabla f(x)$ corresponding to morphogen M is otherwise $\langle \nabla f(x) \rangle = \text{GRADIENT}(M)$, then its adjusted SPH estimate is

$$\langle \nabla f(x) \rangle = \text{GRADIENT}(M) - \text{SECRATE}(M) \times \text{CALIB}(M, s)_{\text{GRAD}},$$

which removes the spurious gradient introduced by an agent's distorted MSF.

How do we know when, during the calibration procedure, the morphogen around an agent has reached steady state? In practice, it may be most practical simply to stop when no further significant change can be detected in some reasonable time period. But the theoretical answer is useful in simulation and may be useful in practice as well.

To answer the above question, we imagine that the agent has been traveling and producing morphogen for an infinite period before the calibration run actually began. Steady state will effectively have been reached when the remaining morphogen produced prior to the beginning of the run is negligible compared to all extant morphogen. If we perform a preliminary run in which the agent emits a single pulse of morphogen at the beginning and then maintains a record of sensor readings, we can estimate the time at which steady state is reached in a regular calibration run.

For an agent that has always been moving at some constant velocity, let $\phi(t)$ be the concentration of morphogen still detectable at time t due to an instantaneous pulse of arbitrary size released at time $t = 0$ (that is, a Dirac delta function scaled by some arbitrary value). Note that $\int_{t_1}^{t_2} \phi(t) dt$ can then be interpreted as either an integral from t_1 to t_2 of concentration readings after the above

pulse or as the concentration reading at time $t = t_1$ if the agent had been releasing morphogen at some constant rate between time $t = -t_2$ and time $t = 0$. To return to our imaginary scenario, steady state will have effectively been reached at time t_1 if

$$\frac{\int_{t_1}^{\infty} \phi(t) dt}{\int_0^{\infty} \phi(t) dt} \leq \alpha,$$

where α is an arbitrary low proportion (0.01 in my simulator).

For some $t_2 > t_1$, we can also write this as

$$\frac{\int_0^{t_2} \phi(t) dt - \int_0^{t_1} \phi(t) dt + \int_{t_2}^{\infty} \phi(t) dt}{\int_0^{t_2} \phi(t) dt + \int_{t_2}^{\infty} \phi(t) dt} \leq \alpha. \quad (3.19)$$

Immediately after the release of an isolated pulse of morphogen, sensor readings will increase as that morphogen reaches the sensors. However, after this brief transient period, sensor readings due to that pulse will decrease monotonically both because the agent is moving away from the site of release and because the morphogen is being degraded in the environment. Therefore, assuming t_2 is greater than this transient period, modeling the tail of $\phi(t)$ based on exponential decay alone will overestimate readings. Where k is a morphogen's degradation rate,

$$\int_0^{\infty} \phi(t) dt < \int_0^{t_2} \phi(t) dt + \int_{t_2}^{\infty} \phi(t_2) e^{-k(t-t_2)} dt = \int_0^{t_2} \phi(t) dt + \frac{\phi(t_2)}{k}.$$

Therefore, by releasing a pulse of arbitrary size at $t = 0$ and keeping a running table of integrated sensor readings such that $A[t'] = \int_0^{t'} \phi(t) dt$, we have the information we need to find conservative estimates, at time t_2 , of the time t_1 at which steady state is effectively reached. Substituting as needed into eq. (3.19), we have

$$\frac{\int_{t_1}^{\infty} \phi(t) dt}{\int_0^{\infty} \phi(t) dt} < \frac{A[t_2] - A[t_1] + \frac{\phi(t_2)}{k}}{A[t_2] + \frac{\phi(t_2)}{k}} \leq \alpha.$$

Setting the latter two expressions equal and solving for t_1 gives

$$t_1 = A^{-1} \left[(1 - \alpha) \left(A[t_2] + \frac{\phi(t_2)}{k} \right) \right], \quad (3.20)$$

where A^{-1} represents reverse table lookup. Alternatively, we can fix a relationship between t_1 and t_2 and work directly from the latter two expressions in eq. (3.20), simply waiting for the inequality

to become true. This is the approach I take in simulation, with $t_2 = 2t_1$. The larger t_2 is relative to t_1 , the smaller role the exponential approximation plays and the more accurate our estimate.

3.5 Beyond Fluid Dynamics: SPH as a Partial Morphogen Compiler

SPH robotic control generally, and my NSF variant in particular, can be viewed as a compilation technique for a subset of Morphgen code.³ This subset consists of systems of PDEs in which fields and their rates of change are described in terms of these fields’ values and spatial derivatives. SPH, far from relating only to fluid dynamics, provides a general means of estimating densities and spatial derivatives of fields based on point masses and field values at discrete locations. Therefore, one crucial step in transforming such PDE systems into ODE systems for individual agents is to replace references to densities and spatial derivatives with appropriate SPH estimates. For NSF-SPH, each abstract field must also be associated with an NSF channel. Section 3.11 concludes the chapter by elaborating on these and other compilation steps, and the present section discusses the compilation idea more broadly. (I have placed this section in the middle of the chapter because I expect this discussion to clarify sections 3.6 to 3.10, which introduce extensions to the Morphgen language.

In the context of NSF-SPH, compilation introduces a layer of abstraction or virtualization between what may be thought of as Morphgen “application” code and what I envision as a library of physical morphogens. Consider a Morphgen program that specifies a velocity or acceleration field for a tissue, along with several other fields. These other fields may be thought of as representing tissue differentiation, gene expression, or even morphogens in a metaphorical sense. To compile such a program into agent code, all fields other than mass density and motion are treated as agent state, and a unique physical “library” morphogen is associated with each field. (In practice, a programmer might find it convenient to specify these associations, because physical morphogens with different ratios of diffusion to degradation rate can be more robust for implementing different fields. However, such low-level associations are not essential in principle: eq. (3.13) specifies the correct production rate regardless of diffusion and degradation rates.) As described in section 3.3, these physical morphogens implicitly provide the smoothing functions needed for each agent to make SPH estimates of a given field and its derivatives.

³As mentioned in section 2.1, I am ignoring many Morphgen features to focus on a crucial subset comprising PDE descriptions of active tissue. These other features include, for example, an object-oriented mechanism for inheritance of class-like tissue types. Extending a compiler to include these features is, relative to the present work, a more conventional compiler design challenge.

To further illustrate this layer of abstraction, consider a Morphgen application program that specifies what the programmer conceptualizes as a morphogen (for example, if the Morphgen code specifies a diffusion and degradation rate as well as sources and sinks). The SPH compilation process would nonetheless treat this field as a tissue state variable and associate it with a physical morphogen, whose physical parameters are independent of those of the abstract morphogen specified by the application code. This layer of abstraction frees the programmer from the need to find a physical substance with particular parameters. Instead, I envision engineering a small library of physical morphogens that can be repurposed to implement a variety of fields in different Morphgen programs.

As discussed in section 2.4, SPH swarm robotic control has focused on enabling swarms to behave somewhat like fluids, as described by the Navier-Stokes or Euler equations. In keeping with this focus, swarms have been specified with acceleration fields, and other fields have been designed to simulate such physical quantities as energy and pressure. In the context of my current work, I reframe these applications as special cases of SPH compilation of implicit Morphgen-like programs that happen to describe roughly physical fluid-like behavior.

In comparison to past fluid-related SPH swarm robotic control, the present work illustrates some of the potential generality of treating SPH as a compilation technique for Morphgen. Rather than specifying physics-inspired quantities such as energy or pressure, Morphgen allows specification of arbitrary fields useful in solving a range of problems. This distinction is a central contribution of my work, and some of the generality this shift in perspectives allows is demonstrated in chapter 5.

A secondary difference between the present work and traditional SPH swarm robotic control is that I specify a velocity, rather than an acceleration, field. With respect to MSFs in particular, velocity-based motion is a natural fit for low-Reynolds number environments in which motile force is approximately proportional to velocity rather than to acceleration. The implications of low-Reynolds number environments are further discussed in section 4.7. But another implication of velocity- rather than acceleration-based control is that the resulting swarm dynamics have more in common with collective motion such as flocking than with Newtonian fluids [157]. I speculate that this affinity with flocking dynamics may contribute to a relative richness of behavior in comparison with fluid-based dynamics. But I do not develop this idea further; and in any case, agents could in principle integrate or differentiate on their own to achieve, respectively, acceleration-based control where force is proportional to velocity or velocity-based control where force is proportional to acceleration.

As mentioned in section 2.1, Morphgen uses the symbol $\mathfrak{D}$ to denote a general time rate of change, with deliberate ambiguity about whether this rate should be interpreted in discrete or continuous terms. I use this symbol as well, with a key difference. In prior work on Morphgen, $\mathfrak{D}$ is interpreted in the Eulerian frame, at a fixed point in space; that is, in continuous terms, $\mathfrak{D} = \partial/\partial t$. Because of the Lagrangian nature of SPH, I choose to interpret $\mathfrak{D}$ in the Lagrangian frame, from the perspective of a moving agent; in continuous terms, $\mathfrak{D} = D/Dt$, where D/Dt is the Lagrangian or material derivative. Eulerian and Lagrangian derivatives are related by

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla.$$

This choice is natural for a system composed of moving agents, because a Lagrangian derivative implies a straightforward update rule for an agent to follow regardless of its velocity. Furthermore, this choice does not impact the expressiveness of Morphgen; a programmer who wants to express change in quantity ϕ in Eulerian terms may simply add a $\mathbf{u} \cdot \nabla \phi$ term to a given change equation, where $\mathbf{u}$ is the agent velocity field; issues related to such terms are explored in section 3.9. Finally, because $\mathfrak{D}$ is associated with Morphgen, which in the context of this work is used to prescribe active tissue behavior, I use $\mathfrak{D}$ to emphasize that a given equation represents active agent or tissue behavior as opposed to, say, physical laws or other descriptions involving derivatives.

Most of the rest of my work focuses on NSF-SPH, and MSF-SPH in particular. But the idea of SPH as a compiler for Morphgen (or similar PDE-based language) is not unique to NSF-SPH; it can be applied to traditional SPH robotic control as well. The key idea behind extending SPH robotic control to compilation of Morphgen is simply that the spatial derivatives estimated by SPH can be applied to a far wider range of PDEs than those describing Newtonian fluids. The capabilities needed for a swarm to serve as a target of such compilation are thus identical to the capabilities needed for traditional SPH robotic control. Thus, any swarm amenable to traditional SPH robotic control can also serve as a target for SPH-based Morphgen compilation. One minimal set of requirements, for example, is for agents to communicate with their individual neighbors, control their velocities, and perform simple internal computation. Alternatively, agents may communicate with a central server which has knowledge of agents' locations, with this server performing the needed SPH computations and commanding agents' velocities.

The next five sections of this chapter introduce extensions to the basic picture of an SPH-based partial Morphgen compiler presented here. Of these five, the first three add new semantic features

to the Morphgen language, allowing programmers to express ideas that make sense independently of a given implementation. Section 3.6 introduces mixtures of differentiation state, and this section is placed first because these differentiation states interact with features described in later sections. Section 3.7 explores processes analogous to reproduction or apoptosis, and section 3.8 describes options for edge detection. These additions to the expressiveness of Morphgen help further advance SPH control beyond its fluid-inspired focus toward more general morphogenesis-inspired behavior.

Following these three sections that introduce new semantic features are two sections that, like section 3.4 above, have the goal of addressing or correcting for limitations in NSF-SPH. But unlike the corrections presented in this earlier section, these two sections introduce extensions to the Morphgen language. These extensions are analogous to pragmas or compiler directives in traditional programming languages: they do not change the meaning of a Morphgen program proper, and they are specific to NSF-SPH implementations of Morphgen. Section 3.9 introduces an upwind scheme to help stabilize hyperbolic PDEs where they appear in Morphgen programs. Section 3.10 addresses deleterious effects of unsmoothness, loosely defined, in density and velocity fields. The final section of this chapter, section 3.11, incorporates these five extensions in presenting the compilation process in more detail.

3.6 Spatial Mixtures of Properties

For some applications, it is useful if spatially colocated tissue can have dissimilar properties. For example, we might want a group of differentiated cells to move through another group of distinctly differentiated cells, both having derived from the same tissue. While the two groups are colocated, fields representing velocity and differentiation state will be discontinuous in space, and therefore difficult to describe using PDEs as required by Morphgen. The obvious solution would be to define two separate tissues to represent the two differentiation states. However, this requires the differentiation states to be discrete. It also requires differentiation to be expressed in terms of a decrease in one tissue’s density accompanied by an increase in the other’s. Such a description fails to capture the idea that the same tissue is simply changing state, a failure that is inelegant and complicates compilation.

A more general solution is to treat properties as dimensions in a configuration space that includes both physical space and properties, such as differentiation state, of which we might want to describe spatial mixtures. Density or velocity fields could be continuous over such a space even when their

projection onto physical space is not. Some interesting generalizations may be possible if we consider SPH smoothing functions over these configuration spaces. However, any such possibilities are beyond the scope of my work, because NSF's are confined to physical space.

Even with NSF's, SPH can be used to estimate field values and derivatives in the useful case in which overlapping tissues have different differentiation states. This is possible even when differentiation state is treated as a continuous variable. To make these estimates, we treat each differentiation state as its own dimension in configuration space, ranging from zero to one, and we assign a separate communication channel to each dimension. We also assign a separate channel to each property of which we would like to make an SPH estimate, whether value or derivative, in the context of a particular differentiation state. For example, we might want to estimate the gradient of some property of interest, for example “*brightness*,” in tissue in differentiation state A , ignoring other colocated tissue. I will denote this idea by adding the differentiation state as a subscript; e.g., “ $\nabla \textit{brightness}_A$ ”. We can make the estimate with the help of two additional channels: one to represent density of tissue in state A , and one to represent the brightness of tissue in state A . Upon compilation using MSFs, secretion rates for these new morphogens would be found easily: the usual secretion rates for density and brightness would be multiplied by the value for state A , and the density of tissue in state A would be substituted for overall density in the secretion rate for brightness.

Treating differentiation states as dimensions allows for continuous treatment of mixtures of tissue. This idea is sketched in fig. 3.3. In this figure, tissue in two differentiation states are colocated in physical space, represented by a single dimension. Tissue in state A is moving in one direction in physical space, and tissue B in the opposite direction. In physical space the velocity field is discontinuous, but in the larger configuration space it may be continuous. In this larger space, the process of differentiation is interpreted as mass moving outward in one of the differentiation state dimensions.

This feature allowing mixed differentiation states interacts with many of the other extensions to Morphogen described below. I discuss particulars of these interactions in the appropriate sections, but they share common features. The interactions occur whenever the implementation of an extension includes ρ in a calculation. My general approach in these cases is to make a separate such calculation with respect to each differentiation state and, if needed, to combine them in some appropriate way such as summation. Specifically, to adapt a calculation involving ρ to a differentiation state A , I replace each occurrence of ρ with ρ_A , the mass density with respect to state A . If the calculation

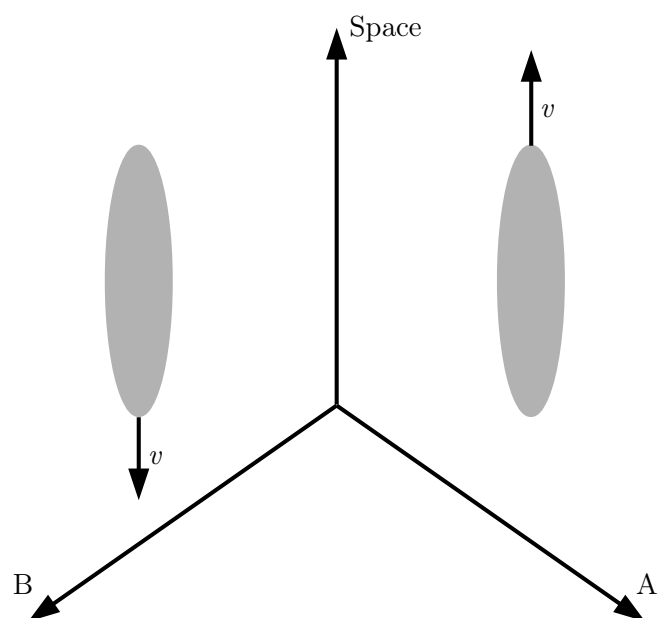


Figure 3.3: Two colocated tissues, distinguished by differentiation states A and B , moving through each other. The velocity field can be made continuous in this configuration space, where each differentiation state is treated as a dimension, even though its projection onto the spatial dimension could not be continuous.

references physical properties of morphogens representing ρ , I replace them with the corresponding properties of the morphogen representing density with respect to state A . I represent this by replacing, for example, k with k_A . Finally, I multiply the term involving ρ_A by A , the degree of differentiation for that state.

These modifications are conceptually straightforward, with the exception of this latter multiplication by A . In general, differentiation states have clearest meaning when they are close to one or zero, and in this case, multiplication by A serves to simply include or exclude the sub-calculation relating to state A . My allowance of intermediate differentiation states mainly serves the purpose of providing for continuous dynamics during the differentiation process. Multiplying by A respects this continuity, so that intermediate differentiation states have intermediate effect.

In explaining how various Morphgen extensions interact with mixed differentiation states, it is more elegant to treat ordinary mass density as being with respect to a default differentiation state whose value is always one. That is, if we need to sum (for example) some expression involving A and ρ_A over all differentiation states A , the summation is understood to include the term for which ρ_A is simply ρ and A is 1.

3.7 Growth and Apoptosis

The mass continuity equation is

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot \mathbf{u}), \quad (3.21)$$

where $\mathbf{u}$ is the physical velocity field of a tissue. ($D\rho/Dt$ is the material or Lagrangian derivative, which represents change in density from the perspective of a parcel of fluid as it moves. It is equivalent to $\partial\rho/\partial t + \mathbf{u} \cdot \nabla\rho$.) Mass continuity is a natural assumption for a Lagrangian method; if particles maintain a fixed mass and are neither created nor destroyed, mass continuity follows. Furthermore, because mass continuity implies that mass density changes only through material motion, assuming mass continuity allows a velocity field to be computed from a purely Eulerian description of a mass density field, and vice versa. However, when we are willing to forgo these possibilities for relating density and velocity (which I do not explore in this work), we can introduce descriptions of tissue for which the continuity equation does not hold. This decouples the density and velocity fields and gives us some ability to describe tissue growth and attenuation.

To describe growth processes in Morphgen, I allow Morphgen programs to contain a special change equation with $D\rho$ as its left-hand side, along with a binary field Γ to signal where and when

it is active.⁴ Such a change equation necessarily represents a break with mass continuity, because under mass continuity the density field is strictly determined by the velocity field. Tissue for which a $\mathcal{D}\rho$ equation is active should therefore implement some form of tissue growth and apoptosis. Where no $\mathcal{D}\rho$ equation is active, mass continuity should continue to hold so that density is naturally determined by velocity.

What should agents do to ensure that a desired $\mathcal{D}\rho$ is satisfied? In an SPH physics simulation, a particle may simply change its mass, but the ability of a physical agent to do so is likely to be limited. Therefore, taking inspiration from biological tissue, I consider processes similar to cellular reproduction and apoptosis (programmed cell death).

As discussed in section 2.5.1, implementation details of NSF-SPH agents are beyond the scope of my work. Similarly, I do not offer thorough accounts of mechanisms by which agents might accomplish something like reproduction or apoptosis. If agents are realized through synthetic biology, they may be able to reproduce as cells normally do, by mitosis. If agents are realized through MEMS technology, recruitment may be more realistic: an agent seeking to “reproduce” may instead recruit from a pool of agents that are inactive in the sense of not yet participating in a Morphogen program. (In something like a hybrid of these approaches, Kriegman et al. [61] describe a replication mechanism in which synthetic multicellular organisms assemble copies of themselves from a pool of loose cells.) In contrast with replication, apoptosis is likely to be straightforward by comparison: If an agent cannot literally destroy itself as a biological cell can, it may simply move away from the tissue or rejoin a pool of inactive agents. I explore reproduction by recruitment in slightly more detail at the end of this section, but otherwise I focus on when, rather than how, an agent should reproduce or apoptose.

I model an agent’s decision to reproduce or apoptose stochastically. Neighboring agents are likely to encounter similar environments, so deterministic decisions may lead to instability. For example, if one agent senses that mass density is too high, it may apoptose. But before the mass density morphogen it had produced has a chance to dissipate and thus inform neighbors of the resulting decrease in local density, those neighbors may make the same decision, and so on. In this way a region of tissue could attenuate more dramatically than desired. Excessive growth may then follow for analogous reasons, leading to oscillations.

⁴Because these extensions are meaningful additions to the Morphgen language and not mere pragmas, I expose Γ to the programmer as though it were an ordinary field. But because it is binary, it only makes sense to set it to zero or to one or to use its value on the right-hand sides of equations, and not to take its spatial derivatives.

An agent should stochastically decide to reproduce or apoptose such that $\mathbb{D}\rho$ at an agent is satisfied in the limit as number density approaches infinity. It would be ideal for this decision to be memoryless, in the sense of an agent's reproducing or apoptosing not depending explicitly on the amount of time since the last such event. Thus we seek a rate λ such that the probability of an agent reproducing (or, if λ is negative, apoptosing) in interval Δt is $\Delta t|\lambda|$ in the limit as Δt approaches zero.

If λ were held steady, it would be the parameter of an exponential distribution describing the time until the next reproduction or apoptosis event for an agent. If in addition the mass velocity field were divergence-free, local density from an agent's perspective would follow an exponential curve:

$$\rho(t) = \rho_0 e^{\lambda t}, \quad (3.22)$$

where ρ_0 is initial density. Taking the derivative (that is, the material derivative, from the agent's perspective) gives

$$\frac{D\rho(t)}{Dt} = \lambda \rho_0 e^{\lambda t}.$$

Comparing with eq. (3.22), we can substitute $\rho(t)$ for $\rho_0 e^{\lambda t}$ where it appears in the derivative:

$$\frac{D\rho(t)}{Dt} = \lambda \rho(t).$$

Now solving for λ gives

$$\lambda = \frac{1}{\rho} \frac{D\rho}{Dt}. \quad (3.23)$$

This result implies that if velocity were divergence-free, we could simply set $\lambda = \mathbb{D}\rho/\rho$. But in general we must account for divergence in the velocity field. If velocity is transmitted as an MSF-SPH vector, an agent can make the needed adjustment directly:

$$\lambda = \nabla \cdot \mathbf{u} + \frac{\mathbb{D}\rho}{\rho}. \quad (3.24)$$

However, this approach requires several additional morphogens as well as orientation sensors (see section 3.1.3), so an alternative approach is preferred. The next two sections present two such alternatives. The designations “integral control” and “proportional control” will be clarified in section 3.7.4.

3.7.1 Integral Control Approach

For a first approach I assume an agent is able to store and retrieve values sampled at runtime, on timescales commensurate with applications. When an agent's value for Γ first becomes true, signalling that $\mathbb{D}\rho$ has become active, the agent begins a running integral of $\mathbb{D}\rho$, using its density estimate at that time, $\langle\rho_0\rangle$, as a starting point. At subsequent times, this integral represents target density, and so the difference between this integral and the current density estimate represents additional density required by the Morphgen program (or, if negative, excess density). That is, additional mass density required at time t is $\langle\rho_0\rangle + \int_{t_0}^t \mathbb{D}\rho(t')dt' - \langle\rho\rangle$.

Dividing this additional mass density required by a fixed time interval b gives us the rate of density change needed to satisfy the Morphgen program by the end of interval b . Substituting into eq. (3.23) tells us how to set λ to achieve this:

$$\lambda = \frac{1}{b\langle\rho\rangle} \left(\langle\rho_0\rangle + \int_{t_0}^t \mathbb{D}\rho(t')dt' - \langle\rho\rangle \right).$$

To maximize the responsiveness of the system, b should be as short as is practical. However, time is needed for the density morphogen produced by a new agent to build up in its environment, or for the density morphogen produced by an apoptosed agent to dissipate. If b is too short, neither the original agent nor its neighbors will have time to register the change in density caused by a reproduction or apoptosis event, and so the probability of another such event shortly afterward will be too high. This bias will lead to oscillations in density for reasons similar to the case of deterministic decisions mentioned previously.

To prevent such oscillations, b should reflect some notion of the time needed for the morphogen field to reach steady state following the reproduction or apoptosis event. As in section 3.4.1, I consider only the total amount of morphogen in the environment, that is, the non-spatial aspect of reaching steady state. Because degradation is assumed to be exponential, the fraction α of morphogen that was present at some initial time that is still present t seconds later is e^{-kt} , where k is degradation rate. Therefore the duration needed to reach some approximation of steady state as defined by a low value of α (currently 0.01 in simulation) is $-\ln(\alpha)/k$. Using this duration for b , we can set λ :

$$\lambda = -\frac{k}{\langle\rho\rangle \ln \alpha} \left(\langle\rho_0\rangle + \int_{t_0}^t \mathbb{D}\rho(t')dt' - \langle\rho\rangle \right). \quad (3.25)$$

3.7.2 Proportional Control Approach

A physically realized agent may lack the memory capacity needed for the above approach, so I next explore an alternative approach to finding λ . Although in general I assume agents are capable of integration, the above approach in particular is a sort of dead reckoning. Neither $\langle \rho_0 \rangle$ nor the integral of $\mathbb{D}\rho$ in eq. (3.25) are tied to any ongoing environment signal. Therefore if an agent's memory of this sum is leaky, the system's fidelity to $\mathbb{D}\rho$ will degrade over time.

As an alternative approach, an agent can use a leaky integral of its mass density estimate to indirectly estimate recent nearby velocity divergence. We can then make use of the general idea of eq. (3.24) in determining λ . However, rather than setting λ , we will derive an appropriate rate of change for λ such that eq. (3.24) is approximately satisfied.

In working toward an estimate of velocity divergence, we first use an agent's leaky integral of its density estimate to estimate recent rate of change in density. By leaky integral, I mean that an agent maintains an internal value B according to

$$\frac{dB}{dt} = \langle \rho \rangle - k_B B, \quad (3.26)$$

where k_B is a decay rate. The “leak” is represented by the term $k_B B$, without which B would be integrated normally. Steady state will occur when $\langle \rho \rangle = k_B B$. When an agent first begins keeping track of this leaky integral, it has little information on which to base an initial value for B , so it may as well assume steady state and initialize B with the value $\langle \rho \rangle / k_B$.

Let us assume that density has been changing at a linear rate and that the agent's estimate of density is accurate, so that $\langle \rho \rangle = a_1 + a_2 t$ for some a_1 and a_2 . Solving for a_2 will then provide the desired approximation of recent $\mathbb{D}\rho/\mathbb{D}t$ in terms of B . Substituting $\langle \rho \rangle = a_1 + a_2 t$ into eq. (3.26) and solving yields

$$B = \frac{a_1}{k_B} + \frac{a_2 t}{k_B} - \frac{a_2}{k_B^2} + c e^{-k_B t}.$$

We assume steady state for B when density is not changing (that is, when $a_2 = 0$), once again for lack of better information. This assumption implies $c = 0$, eliminating the exponential term. Rearranging,

$$a_2 = k_B(a_1 + a_2 t) - k_B^2 B.$$

Now recognizing that $a_1 + a_2 t$ is $\langle \rho \rangle$ and that a_2 is our estimate of recent $D\rho/Dt$, we have the latter in terms known to an agent:

$$\frac{D\rho}{Dt} \approx k_B \langle \rho \rangle - k_B^2 B. \quad (3.27)$$

This estimate accounts for mass density changes due to velocity divergence as well as to nearby reproduction and apoptosis events. It is tempting to think of this estimate as a background rate of density change for which an agent should correct, simply comparing it to $D\rho$ and setting λ accordingly, but this approach is in error. Intuitively, we can think of this erroneous approach as reflecting a collective complacency: If an agent following this approach senses that recent actual change in mass density has been satisfying $D\rho$, it will set λ to zero. However, neighboring agents are likely to behave similarly, causing mass density change to fall behind the demands of $D\rho$.

Instead of making the above mistake, we must keep sight of our goal to realize the idea behind eq. (3.24) by finding an estimate of velocity divergence. If we assume neighboring agents have the same values for λ , we can estimate the portion of total mass density change that is due to velocity divergence specifically. (Because SPH fields tend to be smooth, neighboring agents' values for λ are likely to be similar, so this assumption is justified.) This assumption together with eq. (3.23) imply that the neighborhood rate of mass density change due to reproduction and apoptosis is $\lambda\rho$. Subtracting from our estimate of total mass density change thus estimates mass density change due to velocity divergence alone. Referencing eq. (3.21) for mass continuity, we can now divide by negative density to get velocity divergence itself:

$$\nabla \cdot \mathbf{u} \approx \frac{k_B \langle \rho \rangle - k_B^2 B - \lambda \rho}{-\rho}.$$

Simplifying and replacing ρ with an agent's estimate thereof, we now have a velocity divergence estimate in terms known to an agent:

$$\nabla \cdot \mathbf{u} \approx \lambda - k_B + \frac{k_B^2 B}{\langle \rho \rangle}.$$

We would now like to substitute this divergence estimate into eq. (3.24) (once again replacing ρ there with its NSF-SPH estimate). This substitution results in λ on both the left- and right-hand sides. Therefore, we must interpret the result not as an equality but as an update rule for λ :

$$\lambda_{\text{new}} = \lambda_{\text{old}} - k_B + \frac{k_B^2 B + D\rho}{\langle \rho \rangle} \quad (3.28)$$

or

$$\Delta\lambda = -k_B + \frac{k_B^2 B + \mathbb{D}\rho}{\langle\rho\rangle}.$$

However, instead of updating λ instantaneously, we must give the system time to adjust so that our changes to λ are reflected in $\langle\rho\rangle$. As in the derivation of eq. (3.25), we assume that the longest relevant timescale is the settling duration for the morphogen field representing density, which we approximate as $-\ln(\alpha)/k$ for some small fraction α (currently 0.01 in simulation). Therefore we change λ at the rate that would bring it to the target λ_{new} over this settling period:

$$\mathbb{D}\lambda = \frac{k}{\ln \alpha} \left(k_B - \frac{k_B^2 B + \mathbb{D}\rho}{\langle\rho\rangle} \right). \quad (3.29)$$

I use $\mathbb{D}\lambda$ in place of $D\lambda/Dt$ to emphasize that this change is active agent behavior: the agent integrates $\mathbb{D}\lambda$ to find λ itself. When a $\mathbb{D}\lambda$ equation is first made active, we must choose a value at which to begin integration of $\mathbb{D}\lambda$. By assuming nearby agents have also had the $\mathbb{D}\lambda$ equation switched off, we conclude that the $D\rho/Dt$ estimate from eq. (3.27) now accounts only for velocity divergence and not for reproduction or apoptosis. Therefore we can use eq. (3.28) with λ_{old} set to zero:

$$\lambda_{\text{initial}} = -k_B + \frac{k_B^2 B + \mathbb{D}\rho}{\langle\rho\rangle},$$

Although the above integral requires memory, it is tied to an ongoing environmental signal, so it does not raise the same concerns as the dead reckoning approach discussed in section 3.7.1. For example, if λ drifts too high, the resulting $\langle\rho\rangle$ feeds back into the right-hand side of eq. (3.29), lowering $\mathbb{D}\lambda$ to at least partially compensate.

3.7.3 Interaction with Mixed Differentiation States

For clarity, I have presented these approaches to reproduction and apoptosis in terms of mass density. However, their extension to the differentiation states described in section 3.6 is straightforward. In addition to the $\mathbb{D}\rho$ equation for overall mass density, Morphgen can allow for other optional $\mathbb{D}\rho_A$ equations, where A is a differentiation state. We can compute a distinct λ_A value corresponding to each such equation provided by the user. These λ_A values, along with the original λ corresponding to $\mathbb{D}\rho$, must be combined to find the actual rate that will determine reproduction or apoptosis decisions. One may imagine different ways of combining these various rate parameters into a final λ , but simple summation is appealing. For example, if tissue is undifferentiated, then only $\mathbb{D}\rho$

controls λ . If tissue is differentiated, then $\mathbb{D}\rho_A$ for that differentiation state can be thought of as relative to $\mathbb{D}\rho$ for the overall tissue.

To compute λ_A for a given differentiation state, I make some straightforward changes to eq. (3.25) and eq. (3.29). Every ρ is replaced by ρ_A , the mass density with respect to that differentiation state. Every leaky integral B and its rate k_B are replaced by B_A and k_{B_A} . I also multiply the entire expression by A , the degree of differentiation for that state. (Typically A will be close to either zero or one, but this multiplication provides continuity in case some agents have intermediate differentiation states.) Finally, k , the degradation rate of the morphogen representing mass density, is replaced with the degradation rate of the morphogen corresponding to mass density with respect to differentiation state A , which I here denote k_A . Thus for the integral approach,

$$\lambda_A = -\frac{Ak_A}{\langle \rho_A \rangle \ln \alpha} \left(\langle \rho_{A,0} \rangle + \int_{t_0}^t \mathbb{D}\rho_A(t') dt' - \langle \rho_A \rangle \right),$$

where $\langle \rho_{A,0} \rangle$ is just the initial estimate of ρ_A . Similarly for the proportional control approach,

$$\mathbb{D}\lambda_A = \frac{Ak_A}{\ln \alpha} \left(k_{B_A} - \frac{k_{B_A}^2 B_A + \mathbb{D}\rho_A}{\langle \rho_A \rangle} \right).$$

Finally, after integrating any $\mathbb{D}\lambda_A$ equations in the latter approach,

$$\lambda = \sum_A \lambda_A.$$

(The original λ corresponding to $\mathbb{D}\rho$ is, as described at the end of section 3.6, considered implicit among the λ_A .)

3.7.4 Control Theory

In the context of control theory, and considering rate of density change as the relevant system variable, section 3.7.1 and section 3.7.2 describe integral and proportional control, respectively. This observation suggests tradeoffs between the two approaches. The integral approach is likely to do a better job of maintaining $\mathbb{D}\rho$ over time, at the potential cost of oscillations between overaggressive reproduction and apoptosis. The proportional approach may avoid such oscillations, but may be more likely to generate λ values that are too aggressive or too passive over longer periods of time. This control theory perspective also suggests that, if technology allows, a combination of the two

approaches may be better than either alone, as in PI (proportional-integral) control. Future work may explore these possibilities in detail.

3.7.5 Reproduction By Recruitment

Two complications are introduced when a new agent is created or recruited. First, the new agent may need to adjust for a transitory period before its own morphogen fields have had time to build up to steady state. Second, if the integral approach in section 3.7.1 is used, it may need to adjust its target density to account for a slightly different position in the swarm of agents.

Generally speaking, these complications should be addressed in the context of a specific implementation of reproduction, which I consider beyond the scope of my work. However, I briefly discuss here my approach to these issues in simulation, because they may apply generally to reproduction by recruitment. As discussed further in section 4.10, my simulator does not explicitly model reproduction by recruitment. Rather, it simply places a new agent, instantaneously, near its parent. Nonetheless, this crude model can be viewed as an implicit recruitment model, in the sense that recruitment would lead to a newly recruited agent joining the Morphogen program quickly, near its parent/recruiter.

Under these assumption of fast, proximate recruitment, we can address the two complications mentioned above. I first address the period during which the agent is building up its own morphogen fields toward steady state. Without a correction, an agent would underestimate corresponding field values during this period. As in section 3.4.1, I neglect diffusion time in seeking this correction. For simplicity, I further assume that the agent's morphogen secretion rates are not changing significantly during this building-up period. (This assumption may be relaxed if the agent can maintain a memory of its actual secretion rates, but I do not explore this idea further.)

Under this assumption of constant secretion rate, the mass of morphogen in the environment due to the new agent obeys

$$\frac{d\Phi(L)}{dL} = r(L) - k\Phi(L),$$

where L (for lifetime) is time since the new agent's recruitment and r and k are secretion and degradation rates, respectively. Considering that $\Phi(L) = 0$ at time $L = 0$, the above equation is solved by

$$\Phi(L) = \frac{r}{k} \left(1 - e^{-kL}\right).$$

Dividing $\Phi(L)$ by the steady state solution $\Phi(L) = r/k$ gives the fraction of the steady-state amount present at time L :

$$\frac{\Phi(L)}{\lim_{L \rightarrow \infty} \Phi(L)} = 1 - e^{-kL}.$$

The fraction of the steady-state amount missing at time L is then just e^{-kL} .

As in section 3.4.1, I neglect diffusion time to focus on the non-spatial aspect of morphogen dynamics. This simplification allows us to apply the above fraction of missing morphogen to the concentration of morphogen at points in space near the agent; in particular, at the agent's sensors. From the calibration described in section 3.4.2, the agent can estimate the concentration, per unit secretion rate, that it would sense at steady state given its current speed s . Therefore, following the notation from that section, the additional concentration an agent would sense had it been secreting long enough for its own morphogen contribution to have reached steady state is

$$e^{-kL} \times \text{SECRATE}(M) \times \text{CALIB}(M, s)_{\text{VAL}}.$$

Combining this with the adjustment already described in section 3.4.2, an agent's SPH estimate of the value of the field corresponding to morphogen M is

$$\begin{aligned} \langle f(x) \rangle &= \text{SENSE}(M) \\ &+ \text{SECRATE}(M) \times (\text{CALIB}(M, 0)_{\text{VAL}} - \text{CALIB}(M, s)_{\text{VAL}}) \\ &+ e^{-kL} \times \text{SECRATE}(M) \times \text{CALIB}(M, s)_{\text{VAL}}. \end{aligned}$$

Next we address the need, when the integral approach in section 3.7.1 is used, for an agent to adjust its target density to account for a different position in the swarm of agents. The failure to make such an adjustment may lead to runaway growth resembling tumors. Consider an agent near, but not at, a relatively sharp boundary of the agent swarm. If it recruits a new agent that is more nearly on that boundary, this new agent will estimate a lower mass density for its location, purely by virtue of its location nearer the boundary. But if the old agent simply passes its target density to the new agent, the new agent will compare its lower density estimate to a target density more suited to the interior of the swarm, setting λ too high as a result. The new agent's own recruits are likely to find themselves closer still to the boundary, or beyond it in space otherwise empty of agents, exacerbating the problem.

To address this problem, an agent can pass to a new recruit not only its own target mass density, but also its most recent mass density estimate. The recruited agent, rather than simply adopting its parent’s target density as its own, sets its target density T to the sum of its parent’s target density P and the difference between its initial, uncorrected density morphogen concentration measurement and the parent’s most recent density estimate R :

$$T = P + \text{SENSE}(M) - R,$$

where M is the morphogen corresponding to mass density. This calculation occurs before the recruited agent has secreted any morphogen of its own.

The idea of this correction is that, before the recruited agent has secreted morphogen of its own, its uncorrected measurement of density morphogen reflects the density of its location before its own presence alters that density. Furthermore, the parent’s most recent density estimate similarly reflects density of the parent’s location prior to the new recruit’s presence. The difference between these two values therefore represents the difference in density due to the difference in location per se, abstracted away from the effects of the new recruit’s presence.

The extension of this correction to densities relative to differentiation states is trivial. T , P , and R simply become the target density, parental target density, and parental recent density estimate with respect to a given differentiation state, and M becomes the morphogen corresponding to that same differentiation state.

3.8 Edges

It is often desirable for a Morphgen program to refer to the edge of a tissue. However, it is not obvious how to express the concept of edges in a purely continuous language of PDEs. This difficulty suggests that it may be useful to provide a primitive in the Morphgen language to be specified by an implementation. One option is to convolve mass density with some kernel, looking for locations where the value of this convolution is roughly half the expected mass density of the interior of the tissue [80]. This approach is easy to implement in MSF-SPH, where density estimates are already, and by necessity, convolutions with a kernel. However, this approach requires that we know the expected mass density of the interior of the tissue, and we may want our Morphgen program to remain flexible with respect to such a density.

3.8.1 Mass Density Gradient

One solution relies on mass density gradient, which is already available to an agent. Let us make four simplifying assumptions:

- Mass density gradient in the interior of the tissue is small.
- Agent number density is high relative to smoothing length.
- Curvature of the edge is small relative to smoothing length.
- Agents share a common mass m .

Under these assumptions, we can derive the expected value at an edge for $\|\langle \nabla \rho(\mathbf{x}) \rangle\| / \langle \rho(\mathbf{x}) \rangle$, the ratio of the magnitude of an agent's mass density gradient estimate to its mass density estimate. Making use of eqs. (3.1) and (3.3) for the case of $f(\mathbf{x}) = \rho(\mathbf{x})$, we can write this quantity in terms of the smoothing function evaluated at the distances r_j from an agent on an edge to each neighbor j :

$$\frac{\|\langle \nabla \rho(\mathbf{x}) \rangle\|}{\langle \rho(\mathbf{x}) \rangle} = \frac{\sum_j \frac{m}{\rho_j} \rho_j \nabla W(r_j)}{\sum_j \frac{m}{\rho_j} \rho_j W(r_j)} = \frac{\sum_j \nabla W(r_j)}{\sum_j W(r_j)}.$$

To be more precise about our scenario, consider the half-plane $x \geq 0$ filled with a square grid of agents, with no agents in the other half-plane (fig. 3.4). We are interested in an agent on the edge $x = 0$. Because this scenario is uniform in the y direction, we can work in just one dimension, where gradient becomes an ordinary spatial derivative. And with a as the spacing between agents, the distance r_j from an agent at $x = 0$ to each agent j along our remaining dimension becomes ja :

$$\frac{\sum_j \nabla W(r_j)}{\sum_j W(r_j)} = \frac{\sum_{j=0}^{\infty} W'(ja)}{\sum_{j=0}^{\infty} W(ja)}.$$

By taking the limit as a approaches zero, and by multiplying numerator and denominator by a , we can replace our summations with integrals:

$$\lim_{a \rightarrow 0} \frac{\sum_{j=0}^{\infty} W'(ja)}{\sum_{j=0}^{\infty} W(ja)} = \frac{a}{a} \lim_{a \rightarrow 0} \frac{\sum_{j=0}^{\infty} W'(ja)}{\sum_{j=0}^{\infty} W(ja)} = \lim_{a \rightarrow 0} \frac{\sum_{j=0}^{\infty} W'(ja)a}{\sum_{j=0}^{\infty} W(ja)a} = \frac{\int_0^{\infty} W'(x)dx}{\int_0^{\infty} W(x)dx}.$$

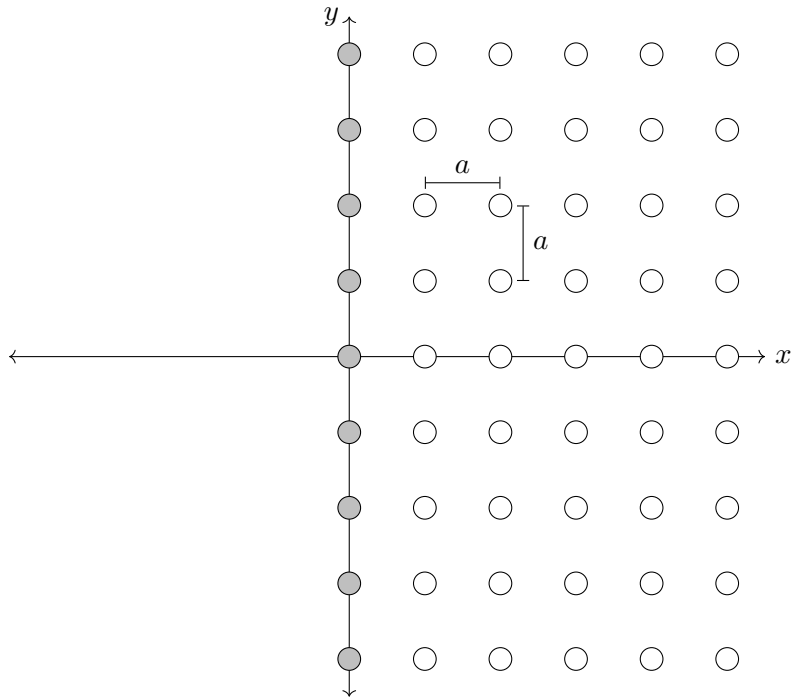


Figure 3.4: A square grid of agents fills the half-plane $x \geq 0$. We derive, in the limit as $a \rightarrow 0$, the ratio of the magnitude of the density gradient to density, as estimated by the shaded agents along $x = 0$. We use this value to determine which agents are on the edge of a tissue.

The integral in the numerator is $W(x)\Big|_0^\infty$ by the fundamental theorem of calculus. Our one-dimensional smoothing function from table 3.2 is $\exp(-r/h)/2h$, so this numerator evaluates to $1/2h$, or, in terms of physical morphogen parameters, $\frac{1}{2}\sqrt{E/k}$. The denominator is $1/2$, because we constructed the smoothing function to have an integral of one over the entire domain. Therefore, given our simplifying assumptions, for an agent at an edge of a tissue, we expect

$$\frac{\|\langle \nabla \rho(\mathbf{x}) \rangle\|}{\langle \rho(\mathbf{x}) \rangle} \approx \sqrt{k/E}.$$

This result also holds for three dimensions, because the analogous half-space grid reduces to the same one-dimensional scenario after considering uniformity in the other two dimensions. In addition, this derivation is the same if we replace mass density ρ with mass density relative to some differentiation state ρ_A , so we can also use this result to detect edges of differentiated regions within tissues.

This derivation allows us to include an edge-finding primitive in our MSF-SPH implementation of Morphgen. We could implement this primitive as a binary function that simply returns `true` when $\|\langle \nabla \rho(\mathbf{x}) \rangle\|/\langle \rho(\mathbf{x}) \rangle$ is greater than some fixed fraction of $\sqrt{k/E}$. However, our result is only a rough approximation, and in practice more flexibility is useful. Therefore I expose to the Morphgen programmer a real number in the range $[0, \infty)$, implemented as the ratio of the left and right sides of the above:

$$\text{EDGEDETECT}() := \frac{\|\langle \nabla \rho \rangle\|}{\langle \rho \rangle} \sqrt{\frac{E}{k}}.$$

This number represents “edge-ness”, with the understanding that the expected value at an idealized edge is one. And if A is an optional differentiation state, we simply subscript ρ with A :

$$\text{EDGEDETECT}(A) := \frac{\|\langle \nabla \rho_A \rangle\|}{\langle \rho_A \rangle} \sqrt{\frac{E}{k}}.$$

3.8.2 Color Fields

An alternative approach to edge detection is to exploit color fields or color functions. Color fields are often used in traditional SPH to calculate surface tension [e.g., 93]. The technique has already been used in traditional SPH robotics to generate behavior emulating physical surface tension [5, 83]. Compared to the approach previously described for edge detection, use of a color field may be more robust to mass density fluctuations within a tissue.

Conceptually, and in its simplest form, a color field has value unity everywhere within a fluid or tissue, and zero everywhere else. To implement this field, we take advantage of the particle

insufficiency property of SPH. (Particle insufficiency is the property that an SPH estimate of any field becomes zero where, in a region that is large relative to smoothing length, there are no particles to support the estimate.) Specifically, we make an SPH estimate of unity; that is, the value of the estimated color field c at point $\mathbf{x}$ is

$$\langle c(\mathbf{x}) \rangle = \sum_j \frac{m}{\rho_j} W(r_j).$$

We can think of this estimate as doing its best to hold the value of one, correcting for mass density fluctuations in the usual manner of SPH. But where there are no particles at all to support it (at least in a region that is large relative to smoothing length), the field drops to zero. Therefore, the gradient of this estimate field remains close to zero wherever there are particles, but becomes large at the edges of a tissue.

The gradient of the color field estimate can be used not only to identify an edge, but also to estimate its curvature, which is needed to compute surface tension. Furthermore, implementing a color field in MSF-SPH is trivial: we simply set a field to unity and assign a morphogen to estimate it. This process may be wrapped in an edge-finding (or even a curvature-of-edge-finding) primitive to be made available to Morphgen code. Alternatively, it may be implemented by the Morphgen programmer directly. The only apparent downside to using a color field in MSF-SPH is the requirement of an additional morphogen.

This technique could also be extended to mixtures of differentiated tissue, at the cost of adding a morphogen for each additional differentiation state. This extension is trivial: a color field with respect to a given differentiation state is just a field again set to unity, this time with respect to that same differentiation state as described in section 3.6. (Traditional color fields use just one field for multiple fluid species, simply using a distinct integer in place of unity for each species. However, this approach makes sense only for distinct interfaces. To accommodate overlapping or mixed differentiation states, a distinct color field is needed for each.)

In informal tests, a color field allows a more accurate identification of edges than using the mass density gradient threshold. In this section, I derive a primitive analogous to `EDGEDETECT` above to detect edges using a color field, although the color field itself must be specified manually by the Morphgen programmer. Future work could move specification of the color field itself to the compiler, as well as further explore the possibility of using color fields to estimate edge curvature.

Our derivation of a new primitive `EDGEDETECTCOLOR` parallels the derivation of `EDGEDETECT` from the previous section. We wish to find the value at a tissue edge, in the continuum limit, of some quantity that relates to the color field and that an agent can calculate, as we did before with respect to the density field. In that case, the value found was for the quantity $\|\langle \nabla \rho(\mathbf{x}) \rangle\|/\langle \rho(\mathbf{x}) \rangle$; in this case we will find a value for $\|\langle \nabla c(\mathbf{x}) \rangle\|$. (As it turns out, the predicted value is the same in both cases: $\sqrt{k/E}$.)

Here it is more convenient to use the form of gradient from eq. (3.2), in which the gradient is outside the summation. From the definition of a color field above, as well as from the prior result $\rho(\mathbf{x}) = \sum_j mW(r_j)$, we have:

$$\|\langle \nabla c(\mathbf{x}) \rangle\| = \nabla \sum_i \frac{m}{\rho_i} W(r_i) = \nabla \sum_i \frac{m}{\sum_j mW_\rho(r_j)} W_c(r_i) = \nabla \sum_i \frac{W_c(r_i)}{\sum_j W_\rho(r_j)}.$$

Because the density and color fields must be represented by distinct morphogens with potentially different physical parameters, their smoothing functions may be different as well, so I label them W_ρ and W_c to keep them separate.

We make the same assumption as previously illustrated in fig. 3.4, of a square grid of agents filling a half-space, with spacing a . As before, we can work in one dimension because of uniformity in the y direction. For the outer summation, the distance r_i between each agent and an arbitrary location x is $|x - ia|$. For the inner summation, the distance from the agent at $x = ia$ to the agent at $x = ja$ is $|i - j|a$:

$$\nabla \sum_i \frac{W_c(r_i)}{\sum_j W_\rho(r_j)} = \frac{d}{dx} \sum_{i=0}^{\infty} \frac{W_c(|x - ia|)}{\sum_{j=0}^{\infty} W_\rho(|i - j|a)}.$$

Once again we take the limit as a approaches zero and multiply by a/a to replace the summations with integrals. We take advantage of the symmetry of an SPH smoothing function in manipulating the limits of these integrals:

$$\begin{aligned} \frac{a}{a} \lim_{a \rightarrow 0} \frac{d}{dx} \sum_{i=0}^{\infty} \frac{W_c(|x - ia|)}{\sum_{j=0}^{\infty} W_\rho(|i - j|a)} &= \frac{d}{dx} \lim_{a \rightarrow 0} \sum_{i=0}^{\infty} \frac{W_c(|x - ia|)a}{\sum_{j=0}^{\infty} W_\rho(|ia - ja|)a} \\ &= \frac{d}{dx} \int_0^{\infty} \frac{W_c(|x - x'|)dx'}{\int_0^{\infty} W_\rho(|x' - x''|)dx''} \\ &= \frac{d}{dx} \int_{-x}^{\infty} \frac{W_c(x')dx'}{\int_{-x'}^{\infty} W_\rho(-x'')dx''}. \end{aligned}$$

Finally, as in the previous section, we apply the fundamental theorem of calculus and evaluate at $x = 0$:

$$\frac{d}{dx} \int_{-x}^{\infty} \frac{W_c(x') dx'}{\int_{-x'}^{\infty} W_\rho(-x'') dx''} = \frac{W_c(x)}{\int_{-x}^{\infty} W_\rho(x'') dx''} \Big|_{x=0} = \frac{W_c(0)}{\int_0^{\infty} W_\rho(x'') dx''} = \sqrt{\frac{k_c}{E_c}},$$

where k_c and E_c are the physical morphogen parameters associated with the color field.

Similar to before, we use this predicted value to create a primitive that hides the details related to physical morphogen parameters but gives the programmer some flexibility on where to draw the threshold defining an edge:

$$\text{EDGEDETECTCOLOR}(c) := \|\langle \nabla c \rangle\| \sqrt{\frac{E_c}{k_c}},$$

As before, the derivation is identical if we replace the general color field with a color field with respect to some differentiation state, so we also have $\text{EDGEDETECTCOLOR}(c_A)$ for some differentiation state A .

3.9 Upwind Scheme for Advection

Upwind schemes help stabilize results in a variety of numerical methods for simulating hyperbolic PDEs such as those describing advective systems. Loosely speaking, they provide an Eulerian simulation with awareness of the direction of motion of a quantity. Consider the basic Eulerian equation for incompressible advection of a quantity ϕ at velocity $\mathbf{u}$:

$$\frac{\partial \phi}{\partial t} = -\mathbf{u} \cdot \nabla \phi.$$

In advection, change in a quantity at some point should logically depend only on the quantity upstream; however, with central differencing alone, change depends on a gradient estimate that looks downstream as much as upstream, resulting in oscillations.

It is not immediately obvious that an upwind scheme is needed for NSF-SPH. In my NSF-SPH implementation of Morphgen, I treat PDEs as referring to a Lagrangian frame by default, which is natural given the Lagrangian nature of SPH. This decision avoids much of the need for upwind methods, because advection can occur through the motion of particles, which carry other field values with them. However, sometimes we wish to reference an Eulerian frame. For example, we may want a pattern to advect across a field of unmoving particles. Conversely, we may want to hold

a pattern steady without regard for the underlying motion of particles. For generality, let's say we want to combine these scenarios: we want to describe advection of some abstract field ϕ according to velocity field $\mathbf{u}_\phi$, without regard for the physical velocity field $\mathbf{u}_a$ of the agents themselves. The corresponding change equation must include an advection term $\mathbf{u} \cdot \nabla \phi$ where $\mathbf{u}$ is $\mathbf{u}_a - \mathbf{u}_\phi$. (The $\mathbf{u}_a$ part of $\mathbf{u}$ can be thought of as providing a sort of “counter-advection” that counteracts the motion of the agents, thus converting between the Lagrangian and Eulerian frames.) Therefore, NSF-SPH does present the challenges that an upwind scheme could address.

Upwind schemes do exist for SPH generally, but there are no obvious ways to implement them using NSFs. Most involve pairwise calculations for which an agent would need information about its individual neighbors [e.g., 59]. At least one other involves an asymmetric smoothing function [14]. Therefore I introduce a novel upwind method for SPH that can be implemented with the limited information available to an agent using NSFs.

Let us consider a particular agent and, for simplicity, place our coordinate system's origin $\mathbf{0}$ at its location. An agent's usual estimate of $\nabla \phi(\mathbf{0})$, based on the corresponding local morphogen gradient for this field, is a type of central difference. Therefore, using this estimate alone in the dot product for the advection term will suffer from the instability that upwinding is intended to mitigate. Our challenge then is to find an estimate of $\mathbf{u} \cdot \nabla \phi(\mathbf{0})$ that gives values of ϕ upwind of an agent (that is, in the $-\mathbf{u}$ direction) more weight than those downwind.

Observe that dividing the above dot product by $\|\mathbf{u}\|$, we obtain the directional derivative, in direction $\mathbf{u}$, of ϕ at the agent's location $\mathbf{0}$:

$$D_{\mathbf{u}}\phi(\mathbf{0}) = \frac{\mathbf{u} \cdot \nabla \phi(\mathbf{0})}{\|\mathbf{u}\|}. \quad (3.30)$$

Let $\mathbf{x}$ be another location in direction $\mathbf{u}$ relative to the agent. (We will specify its choice later.) Then

$$D_{\mathbf{u}}\phi(\mathbf{0}) \stackrel{\text{upwind}}{\approx} \frac{\phi(\mathbf{x}) - \phi(\mathbf{0})}{\|\mathbf{x}\|} \quad (3.31)$$

is a finite-difference estimate, weighted in the upwind direction, of this same directional derivative.

In estimating this directional derivative, an agent can use its internal value of ϕ for $\phi(\mathbf{0})$. For $\phi(\mathbf{x})$, we will use a second-order Maclaurin approximation of $\phi(\mathbf{x})$:

$$\phi(\mathbf{x}) \approx \phi(\mathbf{0}) + \mathbf{x} \cdot \nabla \phi(\mathbf{0}) + \frac{1}{2} \mathbf{x}^\top \mathbf{H}_\phi(\mathbf{0}) \mathbf{x},$$

where $\mathbf{H}_\phi(\mathbf{0})$ is the Hessian matrix of the field ϕ at the agent's location. Unfortunately, I know of no way for an agent to approximate each entry in $\mathbf{H}_\phi(\mathbf{0})$ from the information available to it. However, neither do we know nothing about it: In section 3.1.2 we derive an NSF-SPH estimate of the Laplacian of ϕ , which is the trace of its Hessian. If we make an assumption about the shape of field ϕ , we can construct the Hessian to be consistent with our estimate of the Laplacian.

For simplicity, let us assume that ϕ , at least in the region of a given agent, is a circular paraboloid: $\phi(\mathbf{x}) = a\|\mathbf{x} - \mathbf{b}\|^2$ for some coefficient a and point $\mathbf{b}$. (We have no reason other than simplicity to make this assumption; we are simply looking for a convenient way to take advantage of our Laplacian estimate.) Then the Hessian of ϕ is just the identity matrix scaled by $2a$, and our Maclaurin approximation becomes

$$\phi(\mathbf{x}) \approx \phi(\mathbf{0}) + \mathbf{x} \cdot \nabla\phi(\mathbf{0}) + a\|\mathbf{x}\|^2.$$

Furthermore, the Laplacian of our circular paraboloid ϕ is everywhere, and particularly at the origin, $\nabla^2\phi(\mathbf{0}) = 2aN$, where N is the number of spatial dimensions. Solving for a , we have:

$$\phi(\mathbf{x}) \approx \phi(\mathbf{0}) + \mathbf{x} \cdot \nabla\phi(\mathbf{0}) + \frac{1}{2N}\nabla^2\phi(\mathbf{0})\|\mathbf{x}\|^2.$$

We are getting closer to an approximation that an agent has enough information to make on its own. An agent can use its usual NSF-SPH approximations for the gradient and Laplacian of the ϕ field. As in section 3.1.2, we denote these approximations with angle brackets:

$$\phi(\mathbf{x}) \approx \phi(\mathbf{0}) + \mathbf{x} \cdot \langle \nabla\phi(\mathbf{0}) \rangle + \frac{1}{2N}\langle \nabla^2\phi(\mathbf{0}) \rangle\|\mathbf{x}\|^2.$$

Moving $\phi(\mathbf{0})$ to the left-hand side, dividing by $\|\mathbf{x}\|$, and substituting into eq. (3.31) gives us

$$D_{\mathbf{u}}\phi(\mathbf{0}) \stackrel{\text{upwind}}{\approx} \frac{\phi(\mathbf{x}) - \phi(\mathbf{0})}{\|\mathbf{x}\|} \approx \frac{\mathbf{x} \cdot \langle \nabla\phi(\mathbf{0}) \rangle}{\|\mathbf{x}\|} + \frac{1}{2N}\langle \nabla^2\phi(\mathbf{0}) \rangle\|\mathbf{x}\|.$$

Because $\mathbf{x}$ and $\mathbf{u}$ have the same direction, we can replace $\mathbf{x}/\|\mathbf{x}\|$ with $\mathbf{u}/\|\mathbf{u}\|$:

$$D_{\mathbf{u}}\phi(\mathbf{0}) \stackrel{\text{upwind}}{\approx} \frac{\mathbf{u} \cdot \langle \nabla\phi(\mathbf{0}) \rangle}{\|\mathbf{u}\|} + \frac{1}{2N}\langle \nabla^2\phi(\mathbf{0}) \rangle\|\mathbf{x}\|.$$

Finally, we multiply by $\|\mathbf{u}\|$ both here and in eq. (3.30) to relate our result so far to the actual quantity we seek:

$$\mathbf{u} \cdot \nabla \phi(\mathbf{0}) \stackrel{\text{upwind}}{\approx} \mathbf{u} \cdot \langle \nabla \phi(\mathbf{0}) \rangle + \frac{1}{2N} \langle \nabla^2 \phi(\mathbf{0}) \rangle \|\mathbf{x}\| \|\mathbf{u}\|.$$

It remains to choose $\mathbf{x}$. Because agents themselves represent spatial discretization in SPH, it is natural to want $\mathbf{x}$ to represent some notion of “the next agent over.” Of course, an agent does not know anything about its individual neighbors, but it does have an NSF-SPH estimate of mass density. If it assumes its neighbors share its own mass m , then it can estimate local number density as $\rho_N \approx \langle \rho \rangle / m$. Furthermore, if it assumes its neighbors are arranged on a square grid aligned with its own heading, then it can guess the distance $\|\mathbf{x}\|$ to its nearest upwind neighbor as $1/\sqrt{\rho_n}$ or $\sqrt{m/\langle \rho \rangle}$:

$$\mathbf{u} \cdot \nabla \phi(\mathbf{0}) \stackrel{\text{upwind}}{\approx} \mathbf{u} \cdot \langle \nabla \phi(\mathbf{0}) \rangle + \frac{1}{2N} \langle \nabla^2 \phi(\mathbf{0}) \rangle \sqrt{\frac{m}{\langle \rho \rangle}} \|\mathbf{u}\|.$$

This last approximates our desired advection term entirely in terms known to an agent. We wrap this approximation in a function `VELDOTGRADUPSTREAM( $\phi, \mathbf{u}$ )`, exposed as a primitive of our Morphgen API. Figure 3.5 sketches, for one dimension, an agent’s upwind calculation in the context of its assumptions about the field ϕ and about its neighbors.

3.10 Maintaining Smoothness

As mentioned in section 3.4.1, SPH control tends to generate smooth fields just as traditional SPH models fields as smooth. This tendency arises because SPH estimates are averaged across neighborhoods, so neighboring agents tend to have similar values for these estimates wherever they appear in the agent’s set of ODEs. However, this tendency is not perfect. Through informal explorations, I have noticed that a lack of smoothness⁵ in the density and velocity fields in particular can be detrimental to accuracy in more complex Morphgen programs. For each field, I briefly discuss how a lack of smoothness may arise and under what conditions it is likely to cause problems. More importantly, I describe extensions to allow the Morphgen programmer to encourage smoothness in these two fields in a scalable way.

⁵I use the term *smoothness* colloquially, not in the sense of having well-defined derivatives. What I mean by smoothness is something like having relatively low magnitude of relatively high-frequency components. But I use the term smoothness to emphasize that I am referring to a loosely defined concept that this section seeks to make precise as needed.

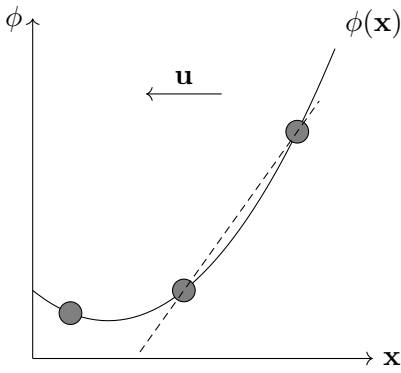


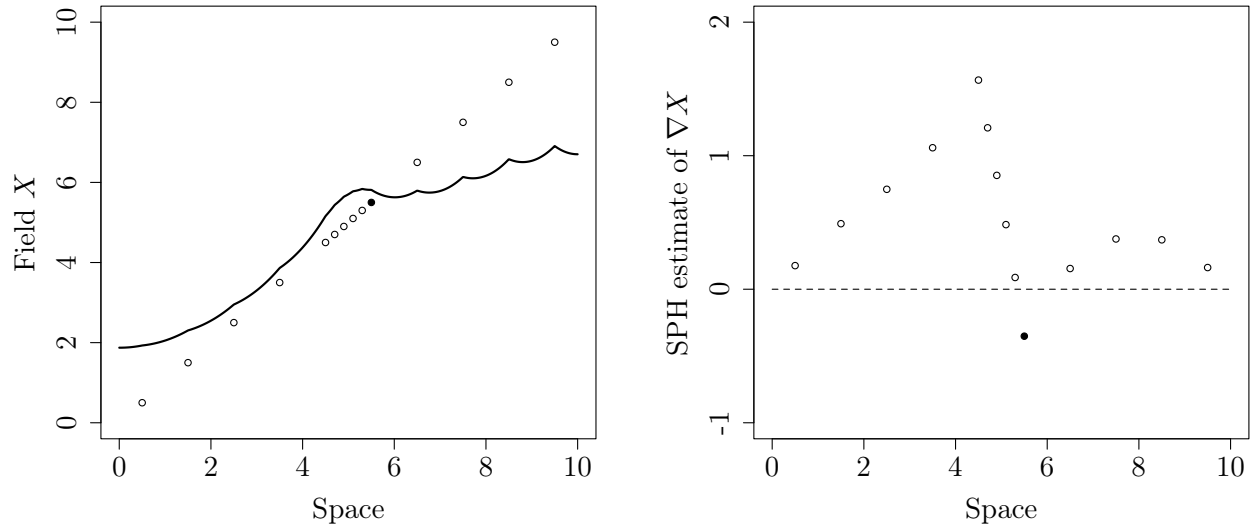
Figure 3.5: A one-dimensional sketch of an NSF-SPH agent's upwind derivative estimate in the context of needed assumptions. The horizontal axis represents space, and the vertical axis represents estimated local values for some field ϕ , assumed to be parabolic. The central circle is the agent making the estimate, and the right and left circles represent the agent's estimate of its nearest neighbors' locations. $\mathbf{u}$ is the direction of advection, which is relative to the agents' physical velocity field. The agent's estimate of the advection term is upwind because it is consistent with the slope of the dashed line connecting its own value of ϕ with its estimate of the value at its nearest neighbor in the $-\mathbf{u}$ direction.

3.10.1 Smoothness of Mass Density Field

An unsmooth mass density field is not necessarily a problem. It does not seem to cause issues with field value estimates or Laplacian estimates, for example. In fact, Morphgen is sufficiently general that a programmer may explicitly encode sharpening behavior (to borrow an image processing term). For example, the one-line Morphgen program $\mathbf{u} = a\nabla\rho$, where a is positive, amplifies unevenness in the mass density field.

On the other hand, unsmooth density fields can interfere with accurate gradient estimates of other fields. For example, a Morphgen program may rely on establishing a smooth gradation in some field, say X , across some region of tissue. In this case, any small source of unsmoothness in density (e.g., uneven initial positions or noisy motion) can cause spurious reversals in SPH gradient estimates of ∇X . Furthermore, if the velocity field has an $a\nabla X$ term for positive a (that is, if agents are to some extent trying to move up the X gradient), small spurious gradient estimate reversals can further decrease density smoothness, exacerbating the problem in a feedback loop. Consider fig. 3.6. I assume agents' values for X successfully form a smooth linear gradation from left to right. Despite this smooth gradation, the unsmooth density field (that is, the clumped agents in the middle) cause a spurious reversal in the NSF-SPH estimate of ∇X . If agents are attempting to move up the X gradient, they should all move rightward, but the spurious reversal will cause the shaded agent to move leftward, amplifying the clumping and exacerbating the problem.

An unsmooth density field can be smoothed as expected, by asking agents to move down the density gradient. A term of the form $-a\nabla\rho$ in the velocity field will suffice for some positive a . Future research may reveal an analytic way to determine what value of a will suffice to keep the density field smooth enough to prevent spurious reversals in the gradient estimates for other fields, but I do not know of one at present. To some extent then, I must leave it to the Morphgen programmer to express the aggressiveness with which smoothness of the density field should be maintained. However, I do not want the programmer to have to write $-a\nabla\rho$ terms directly into Morphgen application code, for two reasons. First, such terms do not describe the ideal behavior of a Morphgen program; rather, they describe a workaround for an imperfect Morphgen implementation, and so should be abstracted away from application code as much as possible. Second, it turns out that the extent to which a program is susceptible to spurious gradient estimates depends on agent mass, which is information available to the compiler but should not be exposed in application-level Morphgen. Therefore I provide a primitive in the form of a field, `DENSITYGUARD`, that the Morphgen programmer can



(a) Internal value of field X for each agent, along with corresponding morphogen field.

(b) MSF-SPH estimate of ∇X for each agent.

Figure 3.6: One-dimensional simulation illustrating a pitfall of unsmooth density fields in estimating gradients of other fields. The circles in each subfigure represent agents. In (a), the vertical position of agents represents their internal value for field X , and the curve represents the concentration of the morphogen field corresponding to field X . In (b), the vertical position of agents represents their estimate of ∇X , based on the difference between the morphogen concentration on either side of their body. The agent is shaded which estimates a spurious reversal of ∇X due to the higher-density clump of agents in the middle. Morphogen fields are given homogeneous Neumann boundary conditions at each end of the domain shown; that is, morphogen is contained by walls at each side.

set in the range $[0, 1)$ to express the aggressiveness with which smoothness should be maintained. Like other Morphogen fields, this smoothness density guard can vary in time and space; however, it is more a pragma than a meaningful part of the Morphogen language proper. It makes sense only to set its value, not to use it on the right-hand sides of equations.

It remains to decide what the compiler should do with this `DENSITYGUARD` value. To this end, we need to more precisely define “unsmoothness” as it relates to the accuracy of field gradient estimates. In informal testing and in two dimensions, the correct sense of unsmoothness seems to be

$$\|\langle \nabla \rho \rangle\| \sqrt{\frac{m}{\langle \rho \rangle}},$$

where m is agent mass. That is, if I ensure (through means described below) that this quantity does not exceed some threshold, then spurious gradient estimate reversals are present or not present regardless of variations in system parameters such as mass density, mass, or physical morphogen parameters.

I do not know a proof that the above quantity is correct, but it is intuitively appealing. The quantity $\sqrt[N]{m/\langle \rho \rangle}$, where N is number of dimensions, is an estimate of interparticle distance (based on an assumption of even spacing). Generalizing to N dimensions, our unsmoothness measure $\|\langle \nabla \rho \rangle\| \sqrt[N]{m/\langle \rho \rangle}$ is therefore a first-order estimate of the change in mass density between one agent and its nearest neighbor in the direction of the density gradient. The inverse of this quantity, $\sqrt[N]{\langle \rho \rangle/m}/\|\langle \nabla \rho \rangle\|$, can thus be seen as a measure of resolution: an estimate of the number of agents along a line segment spanning one unit of mass density change.

Now that we have a definition of unsmoothness, the compiler can add code to counteract it to the extent requested by the programmer through `DENSITYGUARD`. One may imagine various approaches to this task. My approach is to have the compiler construct a new velocity $\mathbf{u}'$ based on the Morphogen program’s requested velocity $\mathbf{u}$ as follows:

$$\mathbf{u}' = \mathbf{u} + \|\mathbf{u}\| \frac{\sqrt[N]{m/\langle \rho \rangle} \langle \nabla \rho \rangle}{\ln(\text{DENSITYGUARD})}. \quad (3.32)$$

(Where `DENSITYGUARD` is zero so that its logarithm is undefined, requested velocity is unmodified.)

This modification can be understood by thinking of $-\ln(\text{DENSITYGUARD})$ as a threshold for unsmoothness that should not be exceeded. Say unsmoothness is exactly equal to this threshold and that the requested velocity is in the direction of the estimated density gradient (that is, threatening to increase unsmoothness beyond the threshold to be allowed). In this case, the two components

of $\mathbf{u}'$ sum to zero and the agent does not move. When unsmoothness is already above threshold, requested velocity may potentially be reversed to regain desired smoothness. When unsmoothness is below threshold, the agent moves more or less as intended, with the component added to its velocity counteracting the density gradient with increasing urgency as threshold is approached.

DENSITYGUARD mechanism is mostly effective in keeping the mass density field smooth enough that agents do not clump together. However, under certain conditions, DENSITYGUARD is not sufficient to break apart small clumps of agents that do occasionally form. Specifically, small clumps can persist when a Morphgen program specifies a tendency for tissue to move up the gradient of some abstract field other than mass density.

My understanding of this problem is incomplete, but it seems to stem from the peak at the center of a typical NSF. Although a real NSF will not have a singularity like an idealized NSF, both its value and gradient become very high near its center. When two agents are very close together, their gradient estimates for all morphogen fields (for mass density or other fields) have strong contributions from the high gradients near the NSF's center. Under these conditions, values for DENSITYGUARD that are otherwise appropriate for a given Morphgen program may not be sufficient to prevent the tendency to move up some field gradient from overwhelming the tendency to move down the density gradient. If DENSITYGUARD is set high enough to fix this, other parts of the tissue may not be able to form reasonable mass density gradients. The same problem can cause agents to hug walls; because an obstacle reflects the diffusion of the agent's own morphogens back to its sensors, this situation is similar to two agents in contact.

To address this problem, I assume that agents are able to sense direct contact with a solid surface. On sensing this contact, they temporarily stop secreting those morphogens representing fields other than mass density. The agent secretes them normally again after a fixed duration. This duration is the usual $-\ln(\alpha)/k$ for morphogen fields to reach steady state, as seen in sections 3.7.1 and 3.7.2. As in those sections, $\alpha = 0.01$ in my simulator. This pause allows agents' tendency to travel down density gradients to bring them away from the other agent or wall before once again seeking to travel up some other gradient.

3.10.2 Smoothness of Velocity Field

Agents' velocity fields can become noisy. Even when agents successfully avoid collisions, neighboring agents may move in dissimilar directions, creating a churning pattern in the tissue. This churn is

detrimental to accurately establishing fields in the tissue, especially smooth gradations of the sort discussed above for which smooth mass density fields are also important.

To a first approximation, neighboring agents experience similar environments and so should make similar decisions, including with regard to velocity. Three effects have the potential to complicate this simplistic expectation.

First, fields whose spatial derivatives do not appear in the Morphgen program are maintained and integrated internally, so their values (and therefore their impact on velocity) may drift apart, even for neighboring agents. In a Morphgen program designed for robustness, such fields will depend, if only indirectly, on sensed values such as environmental cues or fields represented by morphogens, mitigating this issue.

Second, agents may not be able to perfectly realize the velocity vector dictated by the Morphgen code. For example, in my simulations, I assume agents can only move forward, and I model them as turning as they move, as discussed in section 4.7. So although neighboring agents' intended velocities may be similar, their actual velocities may be different because they are, for example, turning in opposite directions to reach similar headings. It is not obvious how to mitigate this contribution to noisy velocity fields. Even if agents wait to turn to their desired heading before beginning to move forward, neighboring agents may still have very different velocities, with some moving forward at full speed and others turning in place.

Third, no morphogen sensor can measure concentration instantaneously. Therefore, if agents move continuously, their sensors are bound to carry, at least in part, information that no longer reflects the agent's current position.

To mitigate this third effect, we can limit agents' speed. It is desirable to give the Morphgen programmer some control over the aggressiveness of this limitation, because unsmooth velocity fields may be more or less detrimental in the context of a given Morphgen program. But once again, the degree to which agents' speed should be limited turns out to depend on details that should not be exposed in the Morphgen language.

In informal testing, it seems that a key factor in the severity of unsmoothness in the velocity field is the extent to which agents with dissimilar velocities pass each other or cross each others' paths, in an informal visual sense. As in section 3.10.1, consider that $\sqrt[N]{m/\langle\rho\rangle}$ is an estimate of interparticle distance in N dimensions. Therefore

$$\|\mathbf{u}\| = \frac{\sqrt[N]{m/\langle\rho\rangle}}{b}$$

is the speed that would bring an agent to a stationary neighbor in duration b . If b is a relevant timescale associated with sensor dynamics, then this speed should give agents time to collect enough information about their changing environment to avoid their crossing paths due to stale state. (For the sensors described in section 4.6, ligand molecules remain bound for τ seconds on average; in this case, I set duration b to $-\tau \ln \alpha$ where α is the arbitrary small fraction 0.01. After this duration, only this fraction of the molecules initially bound remain so.)

There are several crude assumptions involved in the above speed expression, and it is an understatement to say that the idea of agents crossing paths is loosely defined. But this speed is only the basis for a Morphgen primitive allowing the programmer to control the degree of speed limitation. For this reason, the crucial demand is that we capture the correct relationships such that, as relevant system parameters change, a given programmer-determined value for this primitive remains correct with respect to a given Morphgen program.

Similarly to section 3.10.1 above, I introduce a pragma in the form of a field, `VELOCITYGUARD`, once again with range $[0, 1)$. If velocity as modified by eq. (3.32) is $\mathbf{u}'$, then the capped velocity is

$$\mathbf{u}'' = \frac{\mathbf{u}'}{\|\mathbf{u}'\|} \min \left(\|\mathbf{u}'\|, -\ln(\text{VELOCITYGUARD}) \frac{\sqrt[N]{m/\langle \rho \rangle}}{b} \right). \quad (3.33)$$

For zero `VELOCITYGUARD`, speed is not capped, and agents do not move at all as `VELOCITYGUARD` approaches one. The value of `VELOCITYGUARD` for which the speed limit is that derived above is the less-than-intuitive value $1/e$. But again, this particular speed has only tenuous meaning, the important point being the correct scaling of the speed limit with m , $\langle \rho \rangle$, and b .

3.10.3 Interaction with Mixed Differentiation States

Extension of the above to the mixed differentiation states described in section 3.6 is straightforward. In both eq. (3.32) and eq. (3.33), we consider similar equations for each differentiation state A , replacing each ρ with ρ_A and multiplying by degree of differentiation A . Whichever of these equations would most deviate from the original velocity is then chosen; that is, for density smoothness,

$$\mathbf{u}' = \mathbf{u} + \max_A \left(\|\mathbf{u}\| \frac{A \sqrt[N]{m/\langle \rho_A \rangle} \langle \nabla \rho_A \rangle}{\ln(\text{DENSITYGUARD})} \right),$$

and for velocity smoothness,

$$\mathbf{u}'' = \frac{\mathbf{u}'}{\|\mathbf{u}'\|} \min \left(\|\mathbf{u}'\|, \min_A \left(-A \ln(\text{VELOCITYGUARD}) \frac{\sqrt[n]{m/\langle \rho_A \rangle}}{b} \right) \right).$$

(The $\min_A$ and $\max_A$ expressions are understood to include the default differentiation state for which A is one and ρ_A is just ρ , as discussed in section 3.6.)

3.11 Compilation

In section 3.5, I reframed SPH robotic control as the basis for a compiler for Morphgen. In this section, I draw together the various derivations and strategies presented in this chapter to describe what such a compiler should look like. I have two purposes in describing a compiler: first, to summarize and consolidate my ideas, and second, to convince the reader of the practicality of an implementation. That said, it is not necessary for my purposes to write actual code for a compiler. For my simulation, it is most practical to implement my ideas through a combination of compiling by hand and interpretation.

I conceptualize the compilation process as three steps. I present these steps in terms of agents implementing MSFs for clarity, but the process for an ISF model would be similar. Working backward, the final step will target a concrete implementation, real or simulated. The middle step will target an abstraction that will operate in continuous time and which should describe properties that I expect to be common to most potential implementations. The initial step will target a machine that understands the NSF-SPH framework, abstracting away operations shared by all code produced by the middle step and capturing only what is distinct to a given source program. These final, middle, and initial steps are loosely inspired by, respectively, a biological cell, an abstract model of that cell, and an abstract model of its nucleus.

In this presentation, I omit for simplicity the handling of variables representing external fields beyond the control of active tissue, for example the concentration of some environmental marker. Levels and gradients of such fields can of course be included in source change equations, assuming that agents have the ability to sense these quantities along with library morphogens. Their treatment is straightforward but would clutter the code and is not related to the MSF-SPH concepts I wish to illustrate.

I also omit from this presentation the handling of vector fields, again to avoid cluttering the code. As discussed in section 3.1.3, there are different possible ways to map morphogens to vectors. Once a mapping is chosen, constructing and applying the Jacobian is straightforward, but it is messy to spell out in terms of individual morphogens. In addition, I do not yet explore vector fields in simulation.

3.11.1 Stage I: Mapping Variables to Morphogens

In the initial step, the compiler will look at evolutionary PDEs in the source code, and will determine which variables used should be represented by morphogens. Density is always included, because it is needed for other SPH estimates. Other variables are included if their spatial derivatives are used in the source code. For the case in which some PDEs refer to density or derivatives in the context of differentiated states, as described in section 3.6, an additional substance will be assigned for density in the context of that state, and another substance for each non-density variable with a derivative taken in that context.

Each quantity needing a morphogen representation will then be mapped to a distinct physical morphogen. The simplest compiler will make this mapping at random; a marginally better compiler would have a means to accept hints from the programmer. A more sophisticated compiler would analyze the source code for clues as to what physical morphogen properties might be better suited to certain variables, but such an optimization is beyond the scope of this project.

Conceptually, the only other task to be performed at this step is the transformation of evolutionary PDEs from the source code into evolutionary ODEs that each agent can follow. However, there is no practical work for a compiler to do in this regard. Each density estimate and spatial derivative is simply reinterpreted as the appropriate SPH estimate, which my NSF-SPH-aware abstract agent model knows how to make, thanks to the above morphogen mapping. The target agent model for this stage is assumed able to evolve these ODEs over time and to move according to a velocity (or, alternatively, acceleration) function. I also assume it can perform arithmetic needed for the above. It can also secrete morphogens, sense morphogens and their gradients, and divide and apoptose, all as appropriate for the NSF-SPH framework. Algorithm 1 shows this first stage of compilation. In all algorithms shown in this section, each variable or function call inside quotes should be substituted with its value (which may be a number or another expression) if that variable or function call are defined elsewhere among the algorithms. Those not so defined are intended to be passed along unchanged.

Algorithm 1 First step of compilation process, assuming an NSF-SPH-aware agent.

```

morphvars  $\leftarrow \{\rho\}$   $\triangleright$  a set of symbols to be represented by morphogens
for each symbol  $v$  such that a spatial derivative of  $v$  appears in the change equations do
    morphvars  $\leftarrow$  morphvars  $\cup \{v\}$ 
    if  $v$  is subscripted with scalar field symbol  $A$  then
        morphvars  $\leftarrow$  morphvars  $\cup \{\rho_A\}$ 
    end if
end for
Let libmorphs be the list of physical morphogens in the compiler's library
Shuffle libmorphs
i  $\leftarrow 0$ 
for each  $v \in$  morphvars do
    morph[ $v$ ]  $\leftarrow$  libmorphs[ $i$ ]
    Emit “MAPMORPHOGEN( $v$ , morph[ $v$ ])”
    i  $\leftarrow i + 1$ 
end for
Emit all change or motion equations in source (to be reinterpreted as ODEs)

```

3.11.2 Stage II: Implementing NSF-SPH

For my middle step of compilation, I assume an agent model that does not understand NSF-SPH framework, but still has certain other details abstracted away. Specifically, it operates in continuous time, and its body shape, sensor placement, and propulsion mechanisms are left unspecified. It accepts instantaneous velocity (or acceleration) instructions as well as instantaneous secretion rates for each morphogen. It provides continuous sensor estimates for each morphogen’s concentration and gradient. Finally, it accepts instantaneous values for a number such that, were it held constant over time, would be the parameter of an exponential distribution of the duration until a division or apoptosis event. Specifically, if the number is positive, it will be interpreted as referring to a division event, and if negative its absolute value will be interpreted as referring to apoptosis. I continue my assumption from the first step that the agent is able to perform arithmetic and evolve ODEs.

Algorithms 2 to 5 show this middle step of compilation for the minimalist target described above. In these algorithms, E_v and k_v represent the diffusivity and degradation rates of the morphogen associated with field v , and m represents an agent’s mass. A_v represents whatever differentiation state symbol v is subscripted with (that is, defined with respect to, in the sense of section 3.6). In addition, this pseudocode is simplified by assuming that all field variables are subscripted with some differentiation state symbol A ; for those that are not actually subscripted, it is understood that they are with respect to an implicit differentiation state whose value is always one. That is, a field that is not with respect to a differentiation state can be treated as being with respect to a special differentiation state whose value is always one. This simplification makes the pseudocode easier to read by eliminating several conditionals.

Algorithm 2 Helper procedure to compute expression for secretion rate for MSFs. Reflects approach described in section 3.3.

```
procedure SECRATE( $v$ )  
  if  $v$  is of the form  $\rho_A$  then  
    return “ $k_v m A_v$ ”  
  else  
    return “ $k_v m A_v v / \text{ADJVAL}(\rho_{A_v})$ ”  
  end if  
end procedure
```

In presenting this pseudocode, I have focused on summarizing the compilation of code for agents during normal operation, which seems relatively model-independent. I have omitted from this pseudocode some one-time operations such as the initialization of a newly recruited agent or the

Algorithm 3 Helper procedures to compute expressions for value, gradient, and Laplacian estimators, for middle step of compilation.

```

procedure ADJVAL( $v$ ) ▷ sections 3.1.1, 3.4.2 and 3.7.5
  Let  $calibAdj$  be “ $\text{SECRATE}(v) \times (\text{CALIB}(\text{morph}[v], 0)_{\text{VAL}} - \text{CALIB}(\text{morph}[v], |\text{velocity}|)_{\text{VAL}})$ ”
  Let  $lifetimeAdj$  be “ $\exp(-k_v \text{lifetime}) \times \text{SECRATE}(v) \times \text{CALIB}(\text{morph}[v], |\text{velocity}|)_{\text{VAL}}$ ”
  return “ $\text{SENSE}(\text{morph}[v]) + calibAdj + lifetimeAdj$ ”
end procedure
procedure ADJGRAD( $v$ ) ▷ sections 3.1.2 and 3.4.2
  return “ $\text{GRADIENT}(\text{morph}[v]) - \text{SECRATE}(v) \times \text{CALIB}(\text{morph}[v], |\text{velocity}|)_{\text{GRAD}}$ ”
end procedure
procedure LAPLACIAN( $v$ ) ▷ sections 3.1.2 and 3.4.1
  return “ $(k_v/E_v) (g_v + \text{ADJVAL}(v) - A_v v)$ ”
end procedure

```

Algorithm 4 Process source equations for middle step of compilation. Assume everything in algorithm 1 has been done, except for emission of code.

```

for each change or motion equation  $eq$  in source do
  for each VELDOTGRADUPSTREAM( $v, \text{velocity}$ ) in  $eq$  do ▷ section 3.9
    Substitute in its place “ $\text{velocity} \cdot \nabla v + \|\text{velocity}\| \sqrt{m/\rho_{A_v}} \nabla^2 v / 4$ ”
  end for
  for each EDGEDETECT(A) in  $eq$  do ▷ section 3.8.1
    Substitute in its place “ $\|\nabla \rho_A\| \sqrt{E_{\rho_A}/k_{\rho_A}} / \rho_A$ ”
  end for
  for each EDGEDETECTCOLOR( $v$ ) in  $eq$  do ▷ section 3.8.2
    Substitute in its place “ $\|\nabla v\| \sqrt{E_v/k_v}$ ”
  end for
  for each  $v$  where  $v$  is undifferentiated and of the form  $\rho_A$  do
    Substitute ADJVAL( $v$ ) for  $v$  in  $eq$ 
  end for
  for each  $\nabla v$  where  $v$  represents a scalar field do
    Substitute ADJGRAD( $v$ ) for  $\nabla v$  in  $eq$ 
  end for
  for each  $\nabla^2 v$  where  $v$  represents a scalar field do
    Substitute LAPLACIAN( $v$ ) for  $\nabla^2 v$  in  $eq$ 
  end for
  Emit  $eq$ 
end for

```

Algorithm 5 Emit remaining code for middle step of compilation. Assumes velocity control; modification for acceleration control would be straightforward.

```

 $\lambda \leftarrow \text{"0"}$ 
Let  $\alpha$  be an arbitrary small proportion
 $maxVelGrad \leftarrow 0$ 
 $minMaxSpeed \leftarrow \infty$ 
for each  $v \in morphvars$  do
  Emit “SECRETE( $morph[v]$ , SECRATE( $v$ ))”
  if  $v$  is of form  $\rho_A$  then
    Let  $velGrad = \|velocity\| A_v \sqrt{m/ADJVAL(v)} \times ADJGRAD(v) / \ln(DENSITYGUARD_v)$ 
    ▷ section 3.10.1

    if  $\|velGrad\| > \|maxVelGrad\|$  then
       $maxVelGrad \leftarrow velGrad$ 
    end if

    Let  $\tau$  be sensor mean residence time
    Let  $maxSpeed = -A_v \ln(VELOCITYGUARD) \sqrt{m/ADJVAL(v)} / (\tau \ln \alpha)$ 
    ▷ section 3.10.2
    if  $maxSpeed < minMaxSpeed$  then
       $minMaxSpeed \leftarrow maxSpeed$ 
    end if

    if  $\mathbb{D}v$  is specified in source equations then
      if integral control approach is chosen then
        ▷ section 3.7.1
        Emit “ $\mathbb{D}target_{A_v} = \mathbb{D}v$ ”
         $\lambda \leftarrow \lambda$  with “ $+A_v k_{A_v} (target_{A_v} / ADJVAL(v) - 1) / \ln \alpha$ ” appended
      else
        ▷ section 3.7.2
        Let  $B_v$  be the leaky integral associated with  $v$ , and  $k_{B_v}$  its leak rate
        Emit “ $\mathbb{D}B_v = ADJVAL(v) - k_{B_v}$ ”
        Emit “ $\mathbb{D}\lambda_{A_v} = A_v k_{A_v} (k_{B_v} - (k_{B_v}^2 B_v + \mathbb{D}v) / ADJVAL(v)) / \ln \alpha$ ” appended
         $\lambda \leftarrow \lambda$  with “ $+\lambda_{A_v}$ ” appended
      end if
    end if

    else
      Emit “ $\mathbb{D}g_v = A_v \partial v / \partial t - k_v g_v$ ”
      ▷ section 3.4.1
    end if
  end for
  Emit “SETEXPONENTIAL( $\lambda$ )”
  Let  $vel' = velocity + maxGradVel$ 
  if  $minMaxSpeed < vel'$  then
     $vel'' \leftarrow minMaxSpeed(vel' / \|vel'\|)$ 
  else
     $vel'' \leftarrow vel'$ 
  end if
  Emit “SETVELOCITY( $vel''$ )”
  Emit “ $\mathbb{D}lifetime = 1$ ”

```

initialization of growth-related integrals when the binary growth field Γ is switched to its active state. These initialization operations, described in section 3.7 and in chapter 4, are more model-bound (e.g., to the model of reproduction-by-recruitment) than the code for normal operation, which is more general.

This middle stage of compilation summarizes most of the preceding sections of this chapter. This stage primarily involves simple substitutions in source equations and the addition of logic that is mostly independent of source code. No significant source code analysis occurs in this middle step; in fact, unlike the first step, it could be replaced by interpreters in each agent. Such interpreters would turn the agents into the NSF-SPH-aware agents targeted by the first step, and this is the approach I take with my simulator.

3.11.3 Stage III: Targeting Future Technology

The final stage of compilation would target a concrete agent model. As in the second stage of compilation, I assume this concrete agent does not understand NSF-SPH. The agent must be able to evolve the set of ODEs generated by the compilation process. It must be able to move in arbitrary directions. It must be able to secrete or emit fields with the relevant NSF properties. The agent's programmability may range from fully field reprogrammable to not programmable at all. In the latter extreme, a given compiled Morphgen program would be hard-wired at manufacture time.

I do not have enough information about what such a model would look like to write pseudocode for this stage, but I can outline some of the tasks involved. This stage would take into account, for example, the details of sensor locations and how to estimate gradients from their readings. It would also take into account the specific instructions needed to drive a propulsion mechanism, rather than assuming that velocity can simply be dictated. Details of a calibration routine would be addressed in this stage as well. If agents operate in discrete steps, then this stage of compilation would also address time discretization.

In addition, at this stage the more abstract agent's ODE-based controller must be compiled into a target language. This target language may be the machine language of a programmable agent, or it may describe the manufacture of hard-wired agents. I expect agents' control systems to be analog in nature, for power efficiency and because analog control is potentially a good match for ODE-based control. Research suggests translation of an ODE to an analog control system is likely to be feasible. For example, Achour et al. [1] implement a compiler that generates analog device configurations from systems of ODEs.

Translating systems of ODEs into GRNs, for a synthetic morphogenetic implementation, is not yet possible. However, the current state of research into synthetic GRNs suggests that such translation may be feasible in the future. For one thing, ODEs are a common tool for modeling GRNs. (In fact, ODEs are so prevalent in modeling GRNs that some researchers in artificial morphogenesis use “GRN” to refer to controllers based on systems of ODEs [e.g., 137, 85].) Polynikis et al. [118] examine different ODE models of GRNs, comparing the effects of different simplifying assumptions. They argue that improved understanding of GRN models can lead to better ability to synthesize GRNs.

Nielsen et al. [100] are able to compile digital circuits to biological GRNs, which they test successfully in *E. coli*. These circuits’ digital behavior is not closely related to my more analog goal. Nonetheless, the success of their compiler implementation, coupled with the closer relationship that systems of ODEs (as compared with digital circuits) have with GRNs, supports my hope for future compilers from ODEs to GRNs. Other researchers have designed automatic compilers for GRN models at intermediate levels of abstraction, with reference to idealized models of real genes and promoters. One example particularly relevant to my work is that of Beal et al. [7], who compile code written in Proto (see section 2.3.1) into idealized but biologically-plausible GRNs. However, this compiler applies to a non-spatial and digital subset of Proto, limiting its applicability to my inherently spatial and analog framework.

Chapter 4

Simulation Model

I do not attempt to test my ideas using physical agents. As Doursat et al. [22] said, of their category of artificial morphogenesis under which my work would fall, “...most contributions in this category show simulated systems, as large-scale robotic swarms are still too costly to build with today’s technology, and programmable flocking nano-particles are still unheard of.” Attempting to implement my work using synthetic cells is similarly out of reach. Existing robotics simulators would require extensive modification to capture key elements of my model, so I have written a simulator in the C++ language (<https://github.com/allenmcbride/msfsph>).

In the following section I survey the overall control flow of this simulator. In subsequent sections, I explore particular parts of the simulator, explaining some key modeling decisions. Some of these decisions relate to strictly numerical choices, such as spatial discretization; others relate to how I conceive of modeled entities. In both cases, this chapter deals with questions that arise when implementing in simulation the ideas from chapter 3.

A simulation model, such as the one presented here, is important in exploring the plausibility of a future physical implementation of MSF-SPH control. However, this simulation model is not among my research contributions per se, which are described in chapter 3. Instead, it represents one possible numerical simulation involving one possible model of agents that could serve as targets for Stage III of compilation (section 3.11.3).

4.1 Overview of Control Flow

Before setting up the main simulation environment, the simulator checks the availability of calibration data. If the relevant data aren’t found, a calibration routine is run, based on the ideas

in section 3.4.2, before the main simulation environment is set up. For each pair of morphogen and speed, one preliminary run is performed with a pulse to determine the amount of time needed to reach steady state. An arbitrary, fixed number of replicates are then performed and averaged (currently 36 and 64 for crawling and swimming, respectively, but 144 and 192 respectively for zero speed). This averaging abstracts the relevant information from numerical effects related to the finite difference method grid used for diffusion.

The world size for each replicate is based on the ability of the agent to travel long enough for steady state to be reached while ensuring that, at any position, a certain fraction (currently 90%) of the idealized smoothing function mass remains in-bounds. For each replicate, the agent is placed at a random point along the centered circle whose diameter is the needed travel distance, and it travels to the opposite point. At this point, its sensed concentration and gradient are recorded. The set of speeds for which agents are calibrated is arbitrary, but currently starts with $10 \times 10^{-5} \text{ mm s}^{-1}$ and includes each power-of-two multiple of that speed up to $0.65536 \text{ mm s}^{-1}$ or until the number of pixels needed for calibration becomes prohibitively large. The simulator is able to use partial saved calibration data, filling in as necessary (e.g., data found for some speeds or morphogens but not others, or some replicates performed but not enough). When a table lookup is requested by an agent in normal operation, a linear interpolation is returned based on the nearest two calibrated speeds to the agent’s actual speed.

The simulator defines a two-dimensional walled world (one square centimeter by default), possibly containing obstacles or interior walls. The world contains an aqueous medium in which morphogens diffuse and are degraded as discussed in section 3.3. This medium is modeled either considering flow or not, as will be discussed in section 4.11.

To begin, new agents are placed uniformly at random within some hard-coded area of this world. The agents are given some starting state for their field variables. The simulation lets the agents produce morphogen based on these initial values, and diffuses this morphogen, until the morphogen fields are approximately at steady state. During this time the agents do not move or perform nuclear computations. (The amount of time to reach steady state for a given morphogen is approximated as $-\ln(\alpha)/k$, where k is the degradation rate and α is an arbitrary small fraction, currently 0.01.) This procedure, which could be used in real applications as well, simplifies attribution of inaccuracies by avoiding an initial phase in which none of the agents can sense steady-state morphogen fields.

Once this steady state is reached, simulation enters its main loop. At each time step, morphogens are diffused and possibly advected. Each agent is asked in turn to run an update step, which includes

morphogen sensing, internal computation, morphogen production, and motion. If applicable, some agents then reproduce or apoptose. The simulation ends when some fixed time is reached, and various data are saved for analysis. The most important measure of success of a given Morphgen program may relate to a morphogen field, in which case the Euclidean grid storing that field is saved. Perhaps more often, success relates to arrangements of agents; in this case, a kernel density estimate is made with adaptive smoothing length, and the results of this estimate at each cell of a Euclidean grid are saved.

4.2 Overview of Agents

Chapter 3 describes a compilation from Morphgen to agent code in three stages. The pseudocode given in section 3.11 demonstrates that writing this compiler will be straightforward. However, an actual automatic compiler is left to future work. Instead, I perform the first stage of compilation by hand. The simulator then acts as an interpreter for the resulting code, taking the place of the latter two stages of compilation.

From the perspective of an agent, we can think of the simulator in four parts:

- Code produced by the first stage of compilation
- Those aspects of interpretation corresponding to the second stage of compilation (that is, interpretation that realizes the ideas of chapter 3 and therefore defines an abstract MSF-SPH-aware agent)
- Those aspects of interpretation corresponding to the third stage of compilation (that is, interpretation that defines a concrete agent; of course, this being a simulation, “concrete” is a relative term)
- All behavior beyond the control of agent programming (e.g., imperfections in agent control, diffusion and degradation of morphogens, dynamics of the fluid medium, effects of collisions among agents)

We may also collapse the middle two parts into one, defined by all behavior (by which term I include computation) within an agent’s control other than the integration of the code produced by the first stage of compilation:

- Code produced by the first stage of compilation

- All other behavior within an agent’s control, which defines a (relatively) concrete MSF-SPH-aware agent
- All behavior beyond the control of agent programming

This latter view guides my organization of the simulator’s agent-related C++ code, which I divide into three main classes corresponding to the above parts:

- class **Nucleus**, whose public `update()` method contains the code produced by the first stage of compilation
- class **Agent**, which has an object of class **Nucleus** as a private member
- class **AgentManager**, which has an object of class **Agent** as a private member

An **Agent** object can, of course, access only the public methods of its **Nucleus** object. A **Nucleus** object, in turn, contains an object of an **AgentInterface** class whose sole purpose is to expose certain methods of the **Agent** class to **Nucleus**. **AgentInterface** contains a reference to an **Agent** object and is a friend of the **Agent** class. The **AgentManager** and **Agent** classes share an analogous relationship, similarly mediated by public methods and a **ManagerInterface** class.

These class relationships make it clear to the reader that the agent and its metaphorical nucleus have access to only the information they should need for their respective computations. Most importantly, it is clear that the agent lacks any information about its environment other than local morphogen concentrations, and that only as mediated by its sensors. Of the classes related to individual agents, only the **AgentManager** class has access to such information as the agent’s position in the environment or true local morphogen concentrations.

The rest of this chapter roughly follows the order of organization above. First I outline how an agent’s metaphorical nucleus is modeled by the **Nucleus** class, followed by a brief overview of the **Agent** class, which models an agent’s behavior proper. The remaining sections explain modeling decisions relating to agent hardware and the interaction of agents with their environment. Specifically, the **AgentManager** class models shape, sensor hardware, the relation of actual motion to intended motion, collision effects, and hardware failures. Reproduction and apoptosis, as well as fluid effects, are modeled by **AgentManager** as well as other modules that relate to the global simulation environment.

4.3 Nucleus

As discussed in section 3.11.1, the first stage of compilation consists of assigning certain variables to morphogens and reinterpreting PDEs as ODEs by replacing densities and spatial derivatives with NSF-SPH estimates. The first of those tasks is performed by passing the simulator diffusion and degradation parameters for each variable needing a morphogen assignment. The second task also requires little substantive work: A user more or less writes the Morphgen program in C++, directly into the nuclear update loop. Specifically, densities, spatial derivatives, and other primitives described in chapter 3 are replaced with appropriate function calls, and change equations are replaced with explicit increments. For example, the Morphgen line

$$\Delta A = [\rho > 0.1] \nabla^2 A,$$

where A is some field variable and ρ is mass density, becomes

```
fieldVals[A] += dt * findVal(RHO) > 0.1 ? findLaplacian(A) : 0.0;
```

`A` and `RHO` are simply integer indices that specify the location of a particular field variable or density in various data structures. (For clarity, I omit some details in the above code that relate to convenience functions or to aspects of the simulator structure not relevant here.)

Certain information is shared with the rest of the agent through a `NuclearOutput` struct: The most important are desired velocity and the values for each SPH field variable; these are foundational to MSF-SPH. To make differentiation possible (section 3.6), differentiation state variables are shared. To make growth and apoptosis possible (section 3.7), growth rates and boolean growth switches are shared for each differentiation state. To facilitate smoothness of density and velocity (section 3.10), guard values in the range $[0, 1)$ are also shared for each differentiation state.

I do not model agents' failure to faithfully evolve the ODEs that make up a compiled Morphgen program. For example, such failure may occur in the form of leaky integration, faulty timekeeping, or arithmetic that is most accurate within certain ranges of operand values. Introducing models of such failures may be fruitful for future research, especially if they can be associated with likely agent implementations.

4.4 Extranuclear Agent Behavior

The **Agent** class contains all behavior over which an agent has control, apart from that specified by the **Nucleus** class above. This behavior includes that which defines an abstract MSF-SPH-aware agent, as well as that which makes this abstract behavior sufficiently concrete for our chosen level of realism in simulation. There is only one significant examples of the latter category: Although chapter 3 specifies how an abstract MSF-SPH-aware agent corrects morphogen concentration and gradient estimates, it leaves unspecified where the to-be-corrected concentration and gradient estimates come from. The **Agent** class makes the relevant corrections, and it also specifies the relationship between an agent's four sensors and its raw concentration and gradient estimates. Specifically, the raw concentration estimate is the mean of the agent's four sensor readings, and the raw gradient estimate is a simple differencing of morphogen concentrations between sensors in front and back and between sensors to each side, divided by the distances between each pair. In the future, it might make conceptual sense to separate these two abstraction levels into separate classes such as **AbstractAgent** and **ConcreteAgent**, but currently too little code would fit the latter class to justify the added complexity.

The **Agent** class has access only to the information it needs. Constant information about itself is encapsulated in an **AgentParams** struct: the distances between its front and back sensors and between its side sensors, its mass, maximum linear and angular speeds, and the time for its sensors to reach steady state. Other information from the environment is mediated by the **ManagerInterface** class as previously mentioned. In addition to morphogen sensor readings, this information also includes readings from an optional direction sensor, optional boolean sensors for the presence or absence of environmental cues, and booleans indicating whether morphogen sensors have detected enough morphogen molecules to be considered ready for use. Finally, if an agent is an offspring of another agent, it knows this fact, and it knows its parent's internal state at the moment of reproduction.

Because most **Agent** class behavior defines an abstract MSF-SPH-aware agent, this behavior follows closely from the material in chapter 3. Therefore, I leave most further details to the code itself. However, to further the reader's overall sense of the role of class **Agent** in relation to other parts of the simulation, I outline the steps performed in this class's main update routine. This routine is called by the update routine of the corresponding **AgentManager** object. The order of these steps is conceptually arbitrary.

- Update lifetime, for use in reproduction/apoptosis-related calculations.
- Interpret sensor readings. At the moment, this interpretation simply means asking its **AgentManager** for the most recent readings, and if this call returns an invalid reading for any sensor, using the most recent valid value for that sensor.
- Call the update routine of the corresponding **Nucleus** object. Control is inverted during this routine, with the **Nucleus** object able to call **Agent** functions that return various readings and estimates with appropriate corrections.
- Apply smoothness adjustments and upper limits to the ideal velocity vector computed by the **Nucleus** object.
- Update Laplacian correction values.
- Pass the adjusted velocity vector to the `move()` function of its **AgentManager**.
- For each morphogen, pass the appropriate production rate to the `produce()` function of its **AgentManager**.
- Compute the λ value for any differentiation states with the corresponding growth switch set, and return this value to the calling **AgentManager** update routine.

4.5 Agent Geometry

The shape of an agent's body is modeled as an ellipse. Sensors and production instruments are alternately placed around the agent's body at 45° increments of the parameter of the standard parameterization of an ellipse (that is, when the agent's heading is 0° , $\langle x, y \rangle = \langle a \cos(t), b \sin(t) \rangle$, $0 \leq t \leq 2\pi$). This arrangement is shown in fig. 4.1; as shown, the sensors are at the cardinal directions and the production devices between them. For sensors, this placement allows gradients to be estimated with simple differencing, but the appropriate trigonometry would allow for any three or more arbitrarily placed sensors. For production devices, this placement is more arbitrary. Placing the production devices symmetrically front to back and side to side is helpful in making the agent's own expected morphogen fields more symmetric. In principle, I expect calibration to correct for other arrangements, but I have not tested this expectation. Certainly it would make no difference to an agent's behavior if the production devices were placed further toward the ends or middle of the agent.

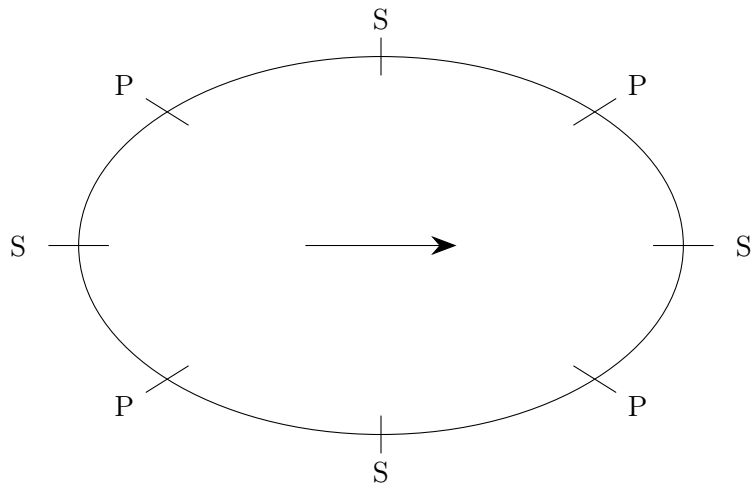


Figure 4.1: Schematic of an agent in simulation. “S” marks sensing devices; “P” marks secretion devices. The arrow indicates the agent’s heading, which is aligned to the major axis of the elliptical body.

For purposes of calculating boundary conditions for morphogen diffusion and for determining when agents or their instruments are in contact, the ellipse shape is simplified. The intersections of the ellipse with grid cell boundaries are computed precisely, but the area of each grid cell occupied by an agent is computed by connecting those intersections with straight lines. The part of each grid cell occupied by an agent is therefore assumed to be a triangle, trapezoid, or pentagon, with one, two, or three right angles, respectively. Collisions and diffusion boundary conditions are discussed further in section 4.8 and section 4.11.2. As also noted in section 4.11.2, agent geometry for fluid medium velocity boundary conditions is simpler still: only the proportion of each partially covered grid cell edge is considered.

4.6 Sensors

I model sensors on a simplified version of sensing in biological cells. A cell’s membrane includes receptors with affinity for molecules of a substance to be sensed. The concentration of this substance is estimated from the number of receptors with associated ligand at a given time. (In biological cells this estimation is mediated by complex signalling cascades, but such detail is beyond the scope of my model.) Engineered sensors more or less following this cell membrane model may be considered types of molecular sensors or biosensors. For our model we assume agents’ sensors will be of this type, though different means of estimating morphogen concentration may be appropriate depending on the nature of a realized agent. For example, in principle it is possible to estimate very low concentrations by measuring individual molecular binding events [132].

In choosing what details to include or omit in my sensing model, I focus on exploring the effects of temporal resolution, bias, and noise at the lower limit of detection. I model discrete ligand binding events, which is both simpler and more realistic than a continuous model with appropriate noise distributions. I assume each sensor has enough binding sites or receptors that we can neglect the decrease in free binding sites with successive binding events. Therefore, I have not systematically explored the effects of limitations on dynamic range. (As mentioned in section 3.3.2, dynamic range can be extended with additional sets of receptors with different binding affinities [154]. My path-finding algorithm (section 5.13), the most demanding of my demonstrations, assumes that a sensor can bind roughly 100,000 morphogen molecules at once without saturating. This number is within an order of magnitude of, for example, the 80,000 cAMP receptors in a typical cell of the *Dictyostelium* genus [145].) In addition to assuming plentiful binding sites, I also assume sufficient numbers of

sufficiently quickly-diffusing morphogen molecules surround an agent that sensor association does not affect ligand availability for additional associations.

Under these assumptions, the rate of future binding events does not depend on the number of previous binding events. Therefore it is natural to model the number of binding events over a given time interval Δt as a sample from a Poisson distribution. The parameter for this distribution will be $\Delta t k c$, where k is the association rate or on-rate determined by the chemistry of receptor and ligand (often denoted k_a or k_{on}), and c is local morphogen concentration. By contrast, in modeling the number of unbinding events over this same time period, the total number of bound ligand molecules may be small enough that we cannot neglect its decrease with successive unbinding events. Therefore we model each bound ligand molecule's potential unbinding as a Bernoulli trial, and the total number of unbinding events in an interval as a sample from a binomial distribution. The number of trials for this distribution is the number N of ligand molecules bound at the beginning of an interval, and the probability for each trial is $1 - \exp(-\Delta t/\tau)$, where τ is mean association (or dwell) time, the reciprocal of dissociation rate or off-rate (k_d or k_{off}). Combining these distributions gives the change in bound ligand molecules over a time interval:

$$\Delta N \sim \text{Poisson}(\Delta t k c) - \text{Binomial}\left(N, 1 - \exp\left(\frac{-\Delta t}{\tau}\right)\right).$$

However, for realistic Δt , and especially near the lower limit of detection, we may care about ligand molecules that both bind and unbind within a given time interval. We consider that molecules newly bound within an interval will be bound on average for at most half that interval's duration. Therefore, we approximate the probability that they dissociate within that same interval as $1 - \exp(-\frac{1}{2}\Delta t/\tau)$. This implies a two-step computation of ΔN :

$$\begin{aligned} \Delta N' &\sim \text{Poisson}(\Delta t k c) \\ \Delta N &\sim \Delta N' - \text{Binomial}\left(N, 1 - \exp\left(-\frac{\Delta t}{\tau}\right)\right) - \text{Binomial}\left(\Delta N', 1 - \exp\left(-\frac{\Delta t}{2\tau}\right)\right). \end{aligned}$$

In the continuum limit, this discrete process obeys

$$\frac{dx}{dt} = kc - \frac{x}{\tau}, \tag{4.1}$$

where x is the continuous analogue of N . Assuming c is constant and solving for steady state gives $c = x/(k\tau)$. Therefore, assuming concentration is changing slowly relative to the sensor's dynamics,

an agent can estimate morphogen concentration at a sensor as

$$c = \frac{N}{k\tau}.$$

I make one refinement to this simple estimate when an agent is newly created, under the assumption that an agent has a timer to measure the interval a that the sensors have been active. Specifically, I maintain the assumption that concentration is changing slowly, but I allow that a sensor has started life with no bound ligand and has not reached steady state. Solving eq. (4.1) and assuming that $x = 0$ at time $t = 0$, we now have

$$x = k\tau c(1 - \exp(-a/\tau)) \quad (4.2)$$

The exponential term approaches zero as a becomes large relative to τ , so this correction is relatively robust to timer inaccuracy. Solving for c and substituting N for x , an agent can make a corrected estimate of morphogen concentration:

$$c = \frac{N}{k\tau(1 - \exp(-a/\tau))}.$$

This correction is useful in helping prevent newly created agents from underestimating density and thus replicating prematurely.

Modeling random individual association and dissociation events captures an important and (at least for this type of sensor) unavoidable source of noise. As mentioned at the beginning of this section, I also want to explore the effect of systematic inaccuracy with each sensor. One can imagine that, for a myriad of reasons, a given sensor might consistently read a little high or a little low. Furthermore, this error might be proportional to concentration; for example, an amplification mechanism might be inaccurate, causing a sensor to return readings that reflect 90% or 110% of the actual number of ligand molecules bound.

To model such error, I assume that each sensor has a fixed associated bias b that relates the true number N of bound ligands to its estimate E of the same: $E = bN$. Prior to simulation, this bias is drawn separately for each sensor from a lognormal distribution the natural logarithm of which has mean zero. This distribution, which has mean 1 and support on $(0, \infty)$, is a natural choice for a multiplicative bias. Because each agent has four sensors, drawing independently for each sensor leads to agents that systematically misestimate both mean concentration as well as gradient.

Therefore, our final model of an agent’s concentration estimate, biased and having corrected for sensor age, is

$$E = \frac{bN}{k\tau (1 - \exp(-a/\tau))}.$$

4.7 Motion

Because Reynolds number is low in the microscale aqueous environment, force is proportional to velocity rather than acceleration [121]. Therefore I assume that agents are able to continually dictate a rotational velocity and speed, rather than an acceleration. However, I model this control as subject to upper limits, two types of bias, and random walk noise. (The simulator also has numerical limits for agents’ speeds, but only in relation to timestep size, which can always be made smaller if needed.)

A physical top linear speed for agents can be set by the user of the simulator, who may want to look to biology for reasonable limits. Among crawling cells, keratocytes are among the fastest; they are wound-healing cells found in many vertebrates, including humans. These cells move up to 45 micrometers per minute in certain fish species [126]. The fastest swimming cells are orders of magnitude faster than the fastest crawlers. The fastest known swimmer is the marine ciliate *Uronychia setigera* at seven millimeters per second [66]. Speeds of several other common cell types are collected by Milo and Phillips [89].

In setting its angular velocity, an agent faces two challenges that can cause oscillations in heading if angular speed is not limited. The first challenge relates to its sensor dynamics, and is inherent to a range of possible NSF-SPH implementations in which sensor dynamics are relatively slow compared to turning speed. An agent’s Morphgen program, as compiled by the NSF-SPH method, determines its desired velocity based in part on sensed information. But because sensors rotate along with the rest of the agent body, the direction of this desired velocity is always relative to the agent’s present heading. Furthermore, sensing is not instantaneous; I model sensor dynamics as essentially those of a leaky integral (section 4.6). Therefore, if the agent turns too quickly, its sensors will not have time to fully reflect its approach to its desired heading. When it reaches what it would ideally recognize as its desired heading, the sensors will retain some stale state, making it think it needs to keep turning. Eventually its sensors will register this overshoot, and the process begins again in the opposite direction.

The second challenge is one I have an incomplete understanding of. This issue applies more specifically to agents that have, as in the present model (section 4.5), morphogen secreting instruments at a few discrete locations around its body. (It would not apply, for example, to a model more like a biological cell with diffuse secretion around the agent.) In this case, the agent’s own MSF field is not rotationally symmetric. If the agent is moving while turning, then its gradient measurements will be distorted by its own rotationally asymmetric MSF field, translated and rotated. This distortion also seems to cause oscillations in heading.

Section 3.4.2 describes a calibration routine designed to correct for gradient distortions due to an agent’s own MSF field. However, this correction only applies to straight-line motion. The calibration process could be extended to incorporate turning while moving, if adding an extra dimension to the calibration lookup table is technically feasible. I have not tested the extent to which such additional calibration might be helpful. It is possible that other dynamics that I do not yet understand also contribute to oscillations in heading.

My current workaround to dampen oscillations in heading is similar for both potential causes: I limit angular speed such that turning occurs on timescales consistent with the dynamics causing oscillations. In the case of sensor dynamics, we can use the duration needed for sensors to approximately equilibrate to a step change in morphogen concentration. For the sensors model described in section 4.6, this duration is $-\tau \ln \alpha$, where α is an arbitrary small fraction, currently 0.01. In the case of the agent’s own MSF field, the relevant duration is, as usual, $-\ln(\alpha)/k$, where k is a morphogen degradation rate.

At compile time, the above durations are computed for sensors and for each morphogen that will be used, and the maximum of these durations is used in limiting angular speed. When turning, the agent continually considers the difference between its current and desired heading, and sets its angular velocity such that it would expect to reach its desired heading at the end of the above duration.

Once an agent’s angular speed is set, there is no clear best choice for a linear speed. Unlike with traditional robotic path planning, an agent does not have a target destination, only an instantaneous desired velocity. For example, let s be the magnitude of its desired velocity. An agent could always set its linear speed to s , or it could set it to s only if it has already reached roughly its desired heading, and zero otherwise.

I choose a third option that seems to work well in practice: The agent considers what position it would be in at the end of a given timestep if it could realize its desired velocity, and treats this

position as a destination. It then chooses the linear speed that will bring its actual position at the end of the timestep as close as possible to this destination. If an agent’s heading is more than 90° away from its desired heading, this speed is clearly zero. Otherwise, it follows from Thales’ Theorem that the needed speed is s multiplied by the cosine of the desired heading change. (To be more precise given discretized time, the cosine is of the angle between the desired heading and what the agent’s heading will be at the end of the given timestep.)

To model imperfections in an agent’s ability to control its motion, I assign each agent a bias relating to linear speed, a bias relating to angular velocity, and to all agents an amount of random walk noise for their motion. The biases, similar to the bias for a sensor described in section 4.6, are drawn for each agent at the beginning of their lives. The bias relating to linear speed is multiplied by the agent’s intended linear speed at each timestep. This bias is just like the bias for a sensor, because once again it is multiplicative: It is drawn from a lognormal distribution, the natural logarithm of which has mean zero. The bias relating to angular velocity is added to an agent’s intended angular velocity at each timestep, so it is drawn from a normal distribution with mean zero.

For random walk noise, one standard deviation is chosen for the entire simulation. For each timestep and for each agent, independent samples for the x - and y -axes are chosen from a normal distribution with this standard deviation and mean zero, and these samples are scaled by the agent’s intended speed. These scaled samples form a vector that is added to the agent’s position. This random walk noise is similar to Brownian motion, but it is not intended to model actual Brownian motion, the effect on agents of which is probably negligible. Rather, this random walk noise is intended as a catchall for any transient sources of randomness in agents’ motion.

4.8 Collisions

In traditional SPH modeling, peaked smoothing functions can help prevent collisions between particles [140]. One can expect then that NSF-SPH, which always has peaks at the centers of its smoothing functions, provides good tools for collision prevention. Although the primary purpose of the “smoothness guard” pragmas described in section 3.10) are to improve accuracy, I have found that suitable settings for these guards also help avoid collisions. However, setting these guards aggressively enough to prevent collisions entirely can inhibit desired collective shape formation. Therefore I include a simple collision model in my simulator.

I assume that agent bodies are soft or robust enough that collisions do not damage them. Because I do not model the effect of external forces on agents (either fluid or rigid-body), simulating collisions must be ad-hoc. My primary goal is to avoid unrealistic logjams when colliding agents have similar intended velocity vectors. I reason that in a more realistic scenario, agents with similar intended velocities would be able to simply push each other out of the way. In addition to avoiding logjams, I also try to keep agents' headings fairly close to their intentions.

I consider an agent to have collided with another agent or obstacle if, after an attempted incremental movement, the proportion of any grid cell it would occupy in its target pose is greater than the free proportion of that grid cell, considering any other agents or obstacles partly occupying that cell. When a collision is observed at a target pose, the simulator tries to find a pose near the target that is unobstructed. To do this, it considers a circle centered on the agent's starting location and passing through the target location. It then picks uniformly at random from the points inside the circle. At the same time, it also picks uniformly at random from all orientations between the starting and target orientations. ("Between" here is in the sense of the shorter way around from one orientation to the other.) This candidate pose is adopted unless it too would result in a collision, in which case the procedure is repeated. If no non-colliding pose is found after some arbitrary number of tries (currently ten), the agent remains at its original pose.

In addition to this model of how agents might move in response to collisions, which is a simplistic stand-in for physics, agents also have an active response to sensing contact with another agent or obstacle. As described in section 3.10.1, after contact, agents will pause secretion of morphogens representing non-density fields for duration $-\ln(0.01)/k$, where k is the slowest degradation rate among morphogens used.

4.9 Hardware Failure

I model hardware failure for agent sensors, propulsion, and production. (Each sensor is modeled as a separate component for this purpose.) All agents share a mean time to failure for each of these components. When an agent is created, a lifespan for each of these components is sampled from an exponential distribution, the parameter of which is the reciprocal of that component's mean time to failure. These hardware failures are modeled in the `AgentManager` class, and the `Agent` class is not aware of them.

After a sensor fails, **AgentManager** reports invalid reading for it. (The **Agent** class currently responds to an invalid reading by continuing to use the most recent valid reading for that sensor, but this behavior is independent of the hardware failure model.) After the propulsion and production components fail, the agent ceases to move or produce, respectively.

4.10 Reproduction and Apoptosis

When an **Agent**'s update function returns a nonzero λ value, the corresponding **AgentManager** samples from a Bernoulli distribution, with success probability of $\Delta t |\lambda|$. As discussed in section 3.7, a successful Bernoulli trial is interpreted as a reproduction event or an apoptosis event depending on whether λ is positive or negative.

In the event of reproduction, the simulator attempts to create a child agent near the parent. It first considers a circle of minimal radius around the agent and a random point on this circle. It attempts to place the child at this point. If it fails because of collision with other agents or obstacles, it attempts several other randomly selected points on the same circle. After a number of failed attempts on this circle, where the number of attempts is proportional to the circumference, it moves on to a larger circle and repeats the process. If no point is found on a circle of some maximum radius (currently ten times the agents' major axis length), the procedure gives up and no new agent is created. In the event of apoptosis, the relevant **AgentManager** is destroyed, along with the corresponding **Agent**.

On reproduction, most internal state of the parent is inherited by the child agent. One exception, in the case of integral control, is parental target density, which is inherited with modification as described in section 3.7.5. In addition, sensor state is not inherited; the child's sensors are initialized with no bound morphogen.

In neither the case of reproduction nor apoptosis is anything special done with the morphogen fields. Although this model of reproduction and apoptosis is crude, the level of implementation detail required for any more mechanistic model would be beyond the scope of this project. A reasonable next future step may be a recruitment model of reproduction. It would be relatively straightforward to implement the recruitment model discussed in section 3.7 by modeling inactive agents with the existing **AgentManager** class.

4.11 Fluid Medium

I consider two simple models of the effect of agents' bodies and motion on the aqueous medium through which morphogens diffuse. The two models take inspiration from motile biological cells, whose locomotion can be crudely divided into crawling and swimming. Even though both models are inspired by three-dimensional scenarios, all my simulations are two-dimensional. Three-dimensional simulations should be a straightforward future extension.

In all simulations, I neglect to model any explicit propulsion mechanism. I treat propulsion as an implementation detail, for two reasons. First, as with biological cells, realized agents may use any of a number of possible propulsion mechanisms. A propulsion implementation might look like a flagellum or like cilia, for example. (See Purcell [121] for an overview of basic mechanisms, Elgeti et al. [24] for an extensive review, and Qiu et al. [123] for another possibility exploiting non-Newtonian fluid properties.) Second, the fluid dynamics implications of a propulsion mechanism would add considerable complexity to the simulator, both in terms of development time and simulation running time. (Mechanistic microswimmers are typically simulated one at a time, with thousands of computational elements in a single agent; e.g., Temel and Yesilyurt [144].)

4.11.1 Crawling

I consider that when an agent is crawling along a solid surface, the fluid medium around it will be relatively less disturbed. Furthermore, morphogen will diffuse in the fluid above the agent (thinking of the surface as underneath the agent) as well as around them. I model this scenario in two dimensions by assuming the fluid does not move and that morphogens diffuse without respect for agents' bodies.

With no fluid velocity field, the only medium-related computation needed is diffusion. I assume constant diffusivity, so that Fick's second law of diffusion simply becomes the heat equation:

$$\frac{\partial \phi}{\partial t} = E \nabla^2 \phi,$$

where E is diffusivity. I model obstacles, as well as the edges of the domain, as walls; therefore, homogeneous Neumann boundary conditions are applied.

I use a simple forward Euler finite difference method (FDM) on a square Cartesian grid. The discrete Laplacian operator uses a nine-point stencil such that its application is convolution with

the kernel

$$\frac{1}{h^2} \begin{bmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{2}{3} & -\frac{10}{3} & \frac{2}{3} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix},$$

where h is the width of a grid cell. I define the boundaries of obstacles and walls as coinciding with grid cell borders, so applying Neumann boundary conditions is simple: Viewing the above convolution equivalently in terms of transfers of morphogen between pairs of neighboring grid cells, we simply avoid any transfer where one of a pair of cells would be out of bounds or part of an obstacle.

This kernel is chosen for its isotropic discretization error [110]. I informally tried to use some anisotropic stencils, such as a standard four-point stencil. Agent motion visibly lacked rotational symmetry despite running the rotationally symmetric Morphgen program shown in section 5.1. This asymmetry surprised me, because such stencils are commonly used, and anisotropy in the macroscopic diffusion is not generally pronounced. However, the number of simulation iterations between when an agent produces morphogen and when that morphogen reaches its own sensors is typically small. I suspect this small number of iterations amplifies micro-scale anisotropy to macro-scale agent behavior, but confirming this hypothesis is not important to my present goals.

Forward Euler methods require small timesteps for stability. If a grid cell with nonzero morphogen is surrounded by grid cells with zero morphogen, solving for the timesteps that prevent negative concentration gives

$$\Delta t \leq \frac{3}{10} h^2.$$

An implicit or semi-implicit method (e.g., Crank-Nicolson) would allow for larger timesteps at the cost of increased complexity. But the forward Euler method has the advantage of simplicity, and smaller timesteps allow for tighter coupling with agent behavior and, once introduced in section 4.11.2, advection of agents' medium.

4.11.2 Swimming

When an agent is swimming, it must displace fluid around its body, inducing a nontrivial velocity field in the fluid. This velocity field represents yet another complication to the ideal morphogen smoothing functions described in table 3.2, raising the question of whether NSF-SPH will still be viable in its presence. Addressing this question fully is beyond the scope of the present work, for

two reasons. First, three-dimensional fluid dynamics are different from those of two dimensions in various (and arguably qualitative) ways. Second, a full accounting of the induced velocity field would require detailing a propulsion mechanism, which I forgo as mentioned previously. Instead, I model in two dimensions the fluid velocity field induced by agent motion, considering not only fluid displacement but also rotational velocity induced by agents' turning.

Reynolds number is close to zero at micrometer scale in an aqueous environment, which means that the effect of viscosity dominates that of inertia [121]. Because inertial forces are relatively unimportant, I model fluid velocity as unsteady Stokes flow, which is simpler than the more general flow described by the full Navier-Stokes equations. For an incompressible fluid with uniform, isotropic viscosity, unsteady Stokes flow satisfies:

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} &= -\frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} \\ \nabla \cdot \mathbf{u} &= 0,\end{aligned}\tag{4.3}$$

where ν is kinematic viscosity. Modeling flow by the steady-state Stokes equations is also a common choice for similar scenarios, despite the fluid velocity changing due to external forces. This choice would be worth exploring in the future, but would entail different solution techniques than those explored here [43].

In part because of this low Reynolds number environment, I do not model the effect of fluid forces on the agents. I assume that fluid dynamics are considered in an agent's design such that, in isolation, it has control over its rotation and speed (though I do model noise and bias in agent motion; see section 4.7). In principle, agents would still be affected by flow caused by neighboring agents, but because of the low Reynolds number, the flow induced by an agent is likely to be significant only very close to its own body [144].

To model fluid velocity, I implement a variation on a simple and efficient method described by Kajishima et al. [50]. No-slip boundary conditions are applied at walls, obstacles, and agent bodies. To model diffusion, I introduce Neumann boundary conditions for the agent body to the method described in section 4.11.1, so that agent bodies are opaque to morphogen diffusion. The interaction of these fluid velocity and diffusion models creates a complication that I resolve by placing agent instruments (sensors and producers) on short stalks to avoid their location in grid cells partly occupied by the agent's own body. These decisions are described in more detail in appendix A.

Chapter 5

Results from Simulation

Having described in chapter 4 a simulated implementation of the ideas in chapter 3, I now use this model to explore the ability of MSF-SPH to accurately implement Morphogen programs. Some of this exploration examines different levels of continuously-valued model parameters such as physical diffusion rates or sensor noise. My primary goal with these quantitative tests is to explore what parameters might be viable at an order-of-magnitude scale, so I generally use logarithmic x -axes. (I do not think enough is currently known about the specifics of future physical MSF-SPH implementations for more precise tests to be useful.) Other exploration in this chapter examines the effect of categorical model elements, for example, comparing results with and without calibration.

Ideally, all these tests would be factorial, to discover unexpected interactions and non-linearities among model elements. But for this work, with a few exceptions I vary one parameter at a time, holding others at what seem to be reasonable values based on informal, preliminary simulations. Except where noted, I model all compiler features that are intended to correct for some deviation from an ideal system, such as those described in section 3.4. For simplicity, and to isolate the effects of changing a given parameter, I do not simulate noise or bias except where otherwise noted. For other parameters, I give my chosen defaults as I introduce each test program.

Ideally, all tests would be performed with both the crawling and swimming models described in sections 4.11.1 and 4.11.2. However, the swimming model, in which agents' bodies displace their fluid medium, involves large systems of equations and is computationally intensive (section 4.11.1 and appendix A). Simulating my crawling model, in which I account for diffusion of morphogen in the medium but not velocity of the medium itself, is much faster. For this reason, I limit most tests in this chapter to the crawling model. I perform a few key comparisons between crawling and

swimming models to demonstrate that they perform similarly; the results of these comparisons are presented in section 5.14. This section is at the end of this chapter, after the intervening sections have introduced the Morphgen programs used in these comparisons.

Throughout this chapter, I sometimes refer to agents and sometimes to tissue. My goal in speaking in terms of one or the other is to refer to tissue in contexts that are meaningful at the continuum level, and to agents in contexts specific to the MSF-SPH implementation.

5.1 A Simple Diffusion Test Program

I use a relatively simple Morphgen program to test the effect of each model parameter. The first and simplest is the following:

substance tissue :

scalar fields :

ρ // Mass density

vector fields :

$\mathbf{V}$ // Tissue velocity

behavior :

$$\mathbf{V} = -0.0001 \frac{\nabla \rho}{\rho}$$

This program evolves the mass density field according to isotropic Fickian diffusion (or, equivalently, the heat equation), with diffusivity $0.0001 \text{ mm}^2 \text{ s}^{-1}$. Before further discussing parameter tests, I make two related observations that may clarify some ideas from chapter 3.

Second, although this Morphgen program is derived from Fick's first law, the resulting motion of agents is unlike the motion of particles diffusing physically. Fick's first law is

$$\mathbf{J} = \alpha \nabla \rho, \tag{5.1}$$

where $\mathbf{J}$ is mass flux and α is diffusivity.¹ This law emerges as a statistical description of large numbers of particles executing random walks with similar speed. Because these particles are assumed to be moving in random directions with respect to their neighbors, their motion is not described by

¹Fick's second law, $\partial \rho / \partial t = \alpha \nabla^2 \rho$, follows from the first with the additional assumption of mass continuity. The heat equation is also equivalent, as long as diffusivity is isotropic and constant.

any continuous velocity field. But SPH assumes continuous (and deterministic) fields, so to derive our Morphgen program we must find a continuous velocity field for particles such that their motion matches Fick’s macroscopic description of diffusion. To do so, we simply assume the existence of such a velocity field $\mathbf{V}$, which would imply that flux is $\mathbf{J} = \rho\mathbf{V}$. Substituting into eq. (5.1) and dividing by ρ gives the form used in our Morphgen program. In the context of physics simulation, this approach was introduced by Degond and Mustieles [18].

This derivation illustrates two key points. First, SPH fields are continuous and deterministic. Second, SPH particles are abstract entities, and their behavior has no necessary connection to the behavior of any physical particles that may compose a corresponding physical system. In fact, SPH and SPH particles would be just as useful in a universe in which matter was truly continuous and not composed of particles at all.

To return to the main topic of parameter tests, this simple Morphgen program is well suited for several tests in this chapter. It is easy to perform an FDM simulation of this program on a high-resolution grid, for comparison with MSF-SPH simulations. And despite its simplicity, it captures some key elements of MSF-SPH Morphgen compilation: it involves moving agents, and it requires SPH estimates of both mass density and its gradient.

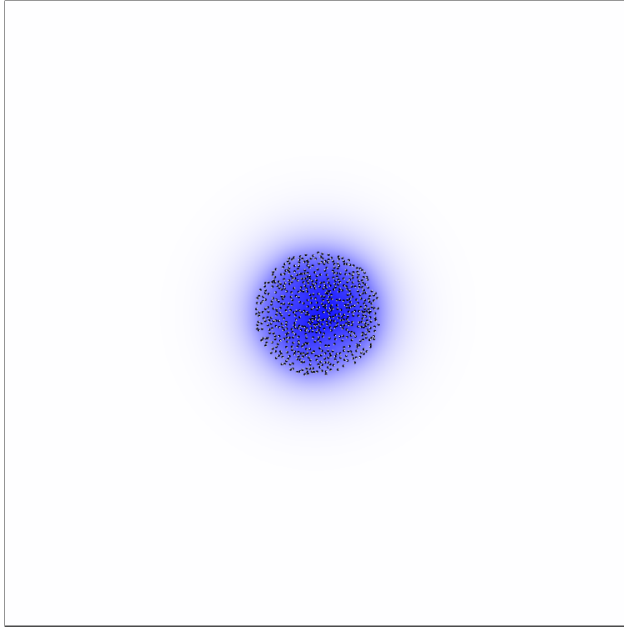
Upon initialization, agents are placed uniformly at random within a disc, 2 mm in diameter, in the center of a square domain 10 mm on a side. Agents continue to be placed, one by one, until relative density² reaches roughly 0.25. Other parameters used are as shown in table 5.1. These parameters are used in all simulations involving this program, except where otherwise noted (for example, when one of these parameters is being varied for a test). Figure 5.1 shows an example of initial and final states for this program, with these default parameters. A video (diffdemo.mp4) is attached, in which one second of video represents one minute of simulation time.

To serve as ideal results for comparison, I use FDM to simulate a continuous version of this Morphgen program, with corresponding initial and boundary conditions. That is, instead of a central disc of randomly placed agents, the simulation starts with a central disc with all pixels assigned density 0.25. I use a grid the same size as a given agent-based simulation to facilitate pixel by pixel comparisons, and I use the maximum stable timestep size. All grids are at least 750×750 pixels, and for this grid size and 0.0001 diffusivity, 54,000 iterations are needed to simulate eight

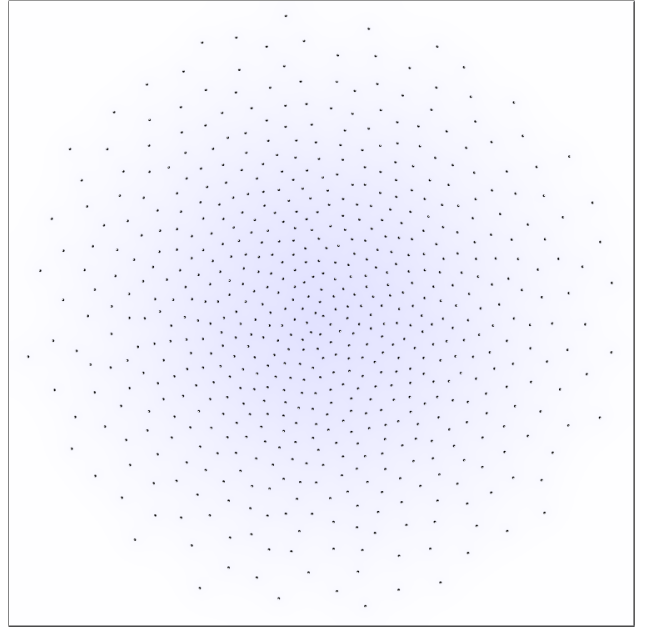
²By *relative density* I mean the overall mass density of agents relative to the density of the fluid medium. But for simplicity, I assume the density of a given agent body is the same as the density of the surrounding fluid, so relative density is really just the proportion of area occupied by agents. In this chapter, when I refer to specific values for mass density, this unitless relative density is what I mean.

Table 5.1: Default parameters for diffusion test program

	Parameter	Value
	Total time	8 h
	Timestep	0.53 s
	Pixels per millimeter in each direction	75
	Agent diameter	40 μm
	Agent body eccentricity	0
	Initial relative density	0.25 (~ 625 agents)
	Morphogen diffusivity	100 $\mu\text{m}^2 \text{s}^{-1}$
	Morphogen half-life	16 second
	Sensor association half-life	16 min



(a) $t = 0 \text{ h}$



(b) $t = 8 \text{ h}$

Figure 5.1: Demonstrating heat equation Morphogen program. Parameters are as given in table 5.1. The morphogen representing the density field is shown in blue. It is scaled so that full saturation occurs at the concentration representing a relative density of 0.25.

hours. The FDM scheme I use is identical to the one described in section 4.11.1 to simulate the diffusion of physical morphogen.

Comparing the results of the FDM and MSF-SPH simulations is not trivial. In the FDM simulation, density is represented by a value at each pixel, but in the MSF-SPH implementation, the mass density field is implicit in the locations of the agents. One option would be to compare, pixel for pixel, FDM results with the morphogen field that represents mass density in an MSF-SPH simulation. The observation that this morphogen field represents a valid SPH estimate of density at all points is, after all, the basis for MSF-SPH. But this estimate is far from an ideal SPH estimate; its kernel (roughly exponential) has a much fatter tail than the gold-standard Gaussian (section 3.3), and it is constrained by the various physical limitations that much of this dissertation is dedicated to addressing.

Instead, I use a traditional SPH estimate based on a Gaussian kernel. Properly speaking, this estimate is a *kernel density estimate* rather than an SPH estimate because it is a static estimate rather than a dynamic simulation. For better accuracy across fields of agents with potentially very uneven density, I use different smoothing lengths for each agent. In two dimensions, a good smoothing function should have about 21 total particles within its support domain [67]. A Gaussian kernel in particular has infinite support domain, so I look for the kernel to have 21 total particles within twice its smoothing length h , which coincides with where many common SPH smoothing functions reach zero. To this end, after a given simulation, I find for each agent the distance to its 20th-closest neighbor, and set that agent’s smoothing length to half this distance. I then make a kernel density estimate for each pixel in the grid.

Having made these kernel density estimates, it is possible to directly compare the mass density results of the FDM and MSF-SPH simulations of our Morphgen program. I use an L^1 norm, taking the absolute value of the difference between each pair of corresponding pixels and then averaging over all pixels. I choose an L^1 over an L^2 norm because I do not wish to be particularly sensitive to outlying data.

5.2 Agent Size and Shape

Scalability is a central motivation for my work in finding compilation techniques for the Morphgen language. By describing tissues and swarms in the continuum limit, one Morphgen program can, in principle, be compiled to target agents of any size, micro- or macroscopic. Of course, agents must be

small relative to the structures and dynamics described by the Morphgen code. In addition, holding mass density constant, one would expect better accuracy from larger numbers of smaller agents, because particle density represents the resolution of discretization for SPH. (This simple expectation is, however, complicated by considerations related to speed as described in section 3.10.2.)

Another central feature of MSF-SPH is the ability to compile a Morphgen program to target given morphogens with different smoothing lengths. The effects on accuracy of diffusivities and degradation rates, which jointly determine smoothing length, are explored further in the next section. For now, because of its importance, I look at accuracy not only for different agent sizes but for different smoothing lengths as well. I vary smoothing length here by varying degradation rate, which is more likely to be controllable than diffusivity (section 3.3.2), while holding diffusivity constant at $100 \mu\text{m}^2 \text{s}^{-1}$.

Figure 5.2 shows accuracy for various agent sizes and morphogen smoothing lengths. Besides agent size and degradation rate, parameters used are as given in table 5.1. With initial relative density held constant at 0.25 across agent sizes, the number of agents ranges from around 10,000 at diameter $10 \mu\text{m}$ to around ten at diameter $320 \mu\text{m}$. Bars in fig. 5.2, and for all subsequent figures in this chapter, represent 95% confidence intervals. At least five simulations are averaged for each data point, which I will indicate with, e.g., $N \geq 5$ in subsequent figures.

As expected, smaller, more numerous agents generally leads to better accuracy, as evidenced by the curves for smaller agents being generally lower than the curves for larger agents. Exceptions occur at the longest smoothing lengths and for the largest agents. I am not sure what explains these unexpected results. They may be an artifact of the simple Morphgen heat equation test program; limitations of this program will be explored more in section 5.4.1. In the case agents of diameter $320 \mu\text{m}$, it seems especially likely that these results may not generalize to other Morphgen programs, because these agents are so large that simulations involve only ten or eleven agents.

Figure 5.2 also shows some evidence that optimal smoothing lengths are shorter for smaller, more numerous agents. This would be an expected result, because in traditional SPH, accuracy is best for certain intermediate numbers of agents within a smoothing function's domain. However, by this reasoning, the optimal smoothing length should scale linearly with agent size, and the apparent differences in optimal smoothing length are not as dramatic as this prediction. More work is needed to explain why optimal smoothing length seems to scale more slowly than expected with agent size. It may be that this expected scaling would hold for more complex Morphgen programs that attempt

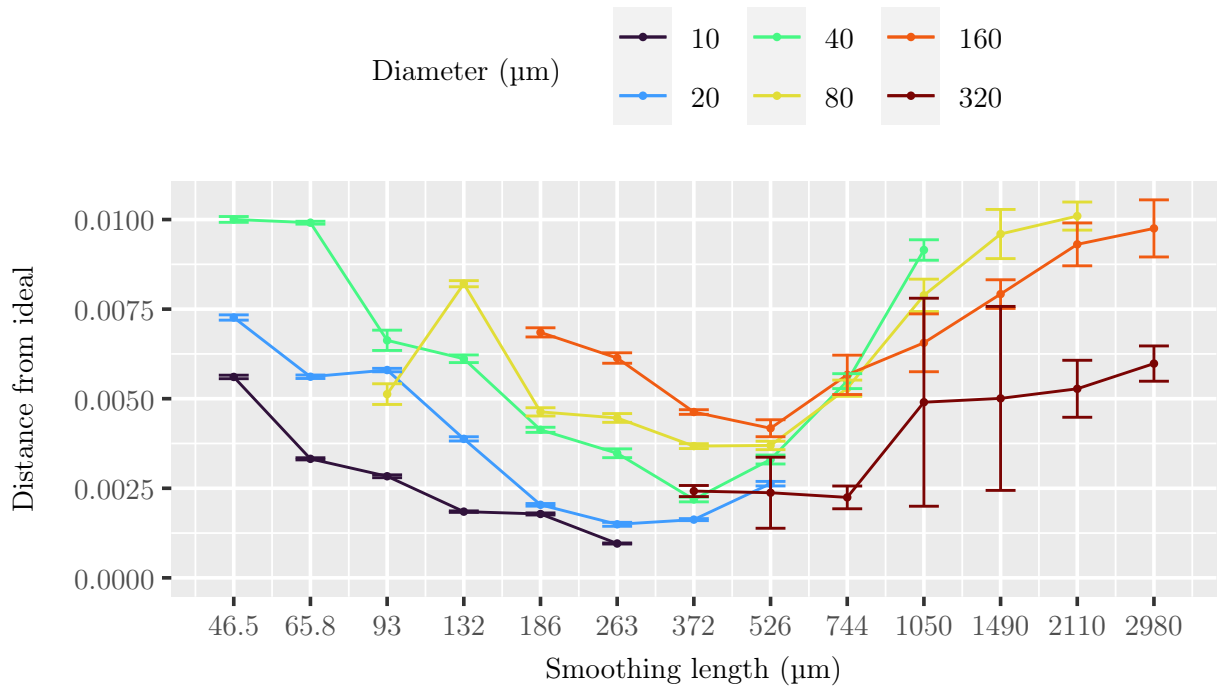


Figure 5.2: Accuracy of heat equation test program by smoothing length, for various agent diameters. Smoothing length is varied by varying morphogen half-life, holding diffusivity constant. Error bars represent 95% confidence intervals. Number of simulations for each data point is at least $N = 5$.

to generate features with a wider range of scales, so that smaller smoothing lengths would allow for the resolution of finer features.

MSF-SPH compilation is, in principle, agnostic to agent shape. For my simulator, a new agent shape requires new code to describe how that shape intersects with the boundaries of the grid cells used to discretize physical morphogen fields. To simplify this code, I limit my exploration of shapes to ellipses of varying eccentricity (section 4.5). Figure 5.3 shows the effect on accuracy of agents of different eccentricity. The agents with zero eccentricity are, as usual, 40 μm in diameter. But for these comparisons, it makes sense to hold agent area constant instead of diameter. So for all three eccentricities here, agent area is about 1257 μm^2 . As expected, changing shape does not have a large effect on accuracy. I am not sure why results are more accuracy for eccentricity 0.8 than zero, but the somewhat less accurate results at eccentricity 0.96 are explained by the high initial density. These agents are so elongated that, when placed randomly in a high-density arrangement, their ends interlock. Most of those placed in the interior of the starting disc, before they can turn toward their intended heading and begin to move outward, must wait for agents closer to the edge of the disc to move away first.

5.3 Physical Morphogen Parameters

The diffusivities E and degradation rates k of physical morphogens are central to the MSF-SPH approach. Their ratio for a given morphogen implies a smoothing length: $h = \sqrt{E/k}$. And their timescales constrain the speed with which a Morphgen program can proceed accurately. To understand what parameter values are likely to be feasible in practice, I perform partially factorial tests of a wide range of parameter combinations. The combinations tested are constrained by my calibration data (section 3.4.2). I have excluded from calibration those combinations with an associated smoothing length that is very long relative to agent size, to avoid excessive computation time. (Excessive computation time is a challenge specifically with simulation; calibration time would not scale with smoothing length in a physical implementation.) I have also excluded those with an associated smoothing length that is very short relative to agent size, to avoid cluttering the figures with information about smoothing lengths that are not likely to be useful.

For these tests, timesteps used are as large as the stability criterion for simulating diffusion via FDM allows, and range from 2.133 s to 0.0667 s for the slowest and fastest diffusivities tested. Apart from timesteps, diffusivity, and half-life, parameters are as given in table 5.1. Results are shown

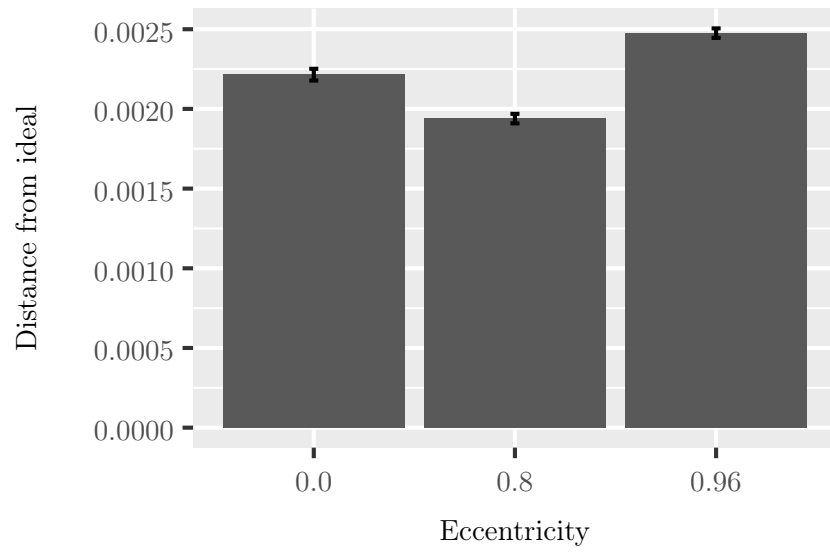


Figure 5.3: Accuracy of heat equation test program for different eccentricities of agents' elliptical body shapes. $N \geq 20$.

in fig. 5.4. Rather than using morphogen degradation rate k directly, I use the more friendly units of half-life $\tau_{1/2}$, in minutes. (Internally, the simulator uses degradation rate in seconds, which is related to half-life by $k = (\ln 2)/(60\tau_{1/2})$.) Figure 5.4a shows accuracy according to half-life (on the x -axis) and diffusivity (represented by different colored curves). Accuracy is best at intermediate half-lives, and the half-lives for which it is best are smaller for higher diffusivity. This result is expected, because diffusivity and half-life jointly determine smoothing length, and SPH is known to be most accurate when smoothing functions cover intermediate numbers of neighbors. (With too few neighbors, not enough information is incorporated in each weighted average, and with too many, higher-frequency features are lost to smoothing.)

To better visualize the relationship between smoothing length and accuracy, fig. 5.4b reorganizes the same data shown in fig. 5.4a. The x -axis is smoothing length, and each colored curve is a different half-life, so that within a given curve, diffusivity controls smoothing length. This view of the data makes clear that, for a given half-life, the optimal diffusivity is that which implies a smoothing length around $400\text{ }\mu\text{m}$. Furthermore, all else equal, accuracy is better at shorter half-lives (and thus also at faster diffusivity, for a given smoothing length). This result is also expected, because faster morphogen dynamics bring the system closer to the idealized assumption in section 3.3.1 that agent speeds are slow relative to morphogen dynamics.

5.4 Agent Imperfections: Noise, Bias, Hardware Failure

5.4.1 Avoiding Spurious Results in Measuring Accuracy

The behavior of real MSF-SPH agents is likely to be far from ideal. Furthermore, the nature of such imperfections is difficult to foresee without knowing what sort of technology will be used in their eventual construction. Sections 4.6, 4.7 and 4.9 describe my efforts to model some likely types of imperfections. In this section I evaluate the impact of different degrees of these types of imperfection on accuracy.

Evaluating the impact of agent imperfections reveals a limitation of the simple Morphogen program used so far in this chapter. Deviations in agent behavior from the ideal tend to make collective agent motion qualitatively noisier and thus more dispersive. If the simple heat equation Morphogen program were perfect when using ideal agents, then any extra dispersive behavior would be measurable as inaccuracy. But the derivation of MSF-SPH makes simplifying assumptions, beyond the assumption of perfect agent behavior, that cannot be met in reality (sections 3.1

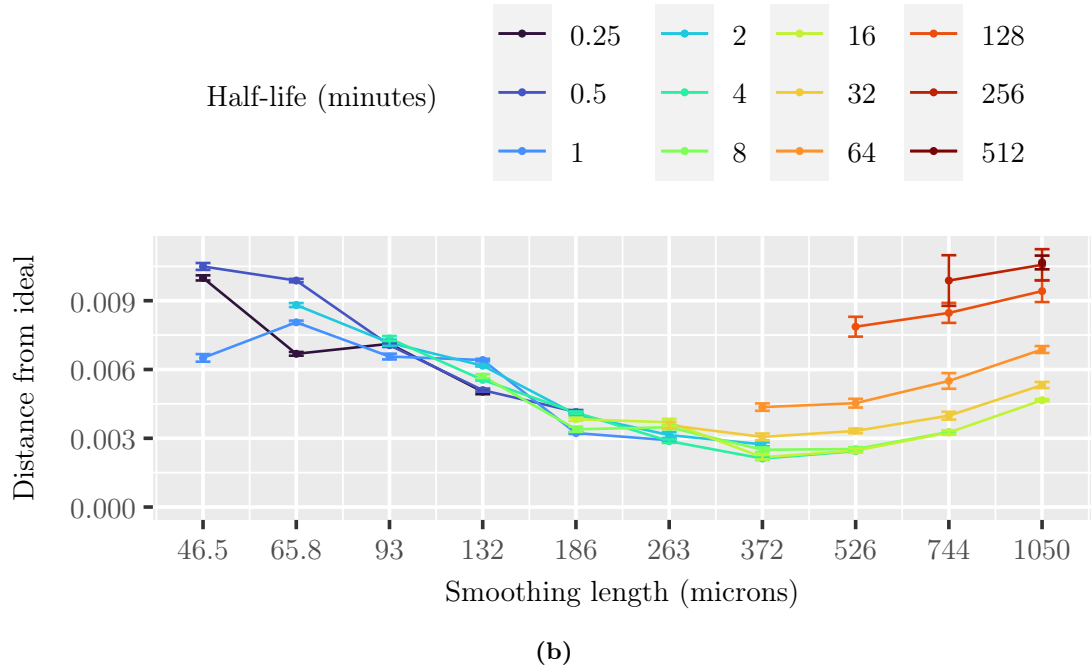
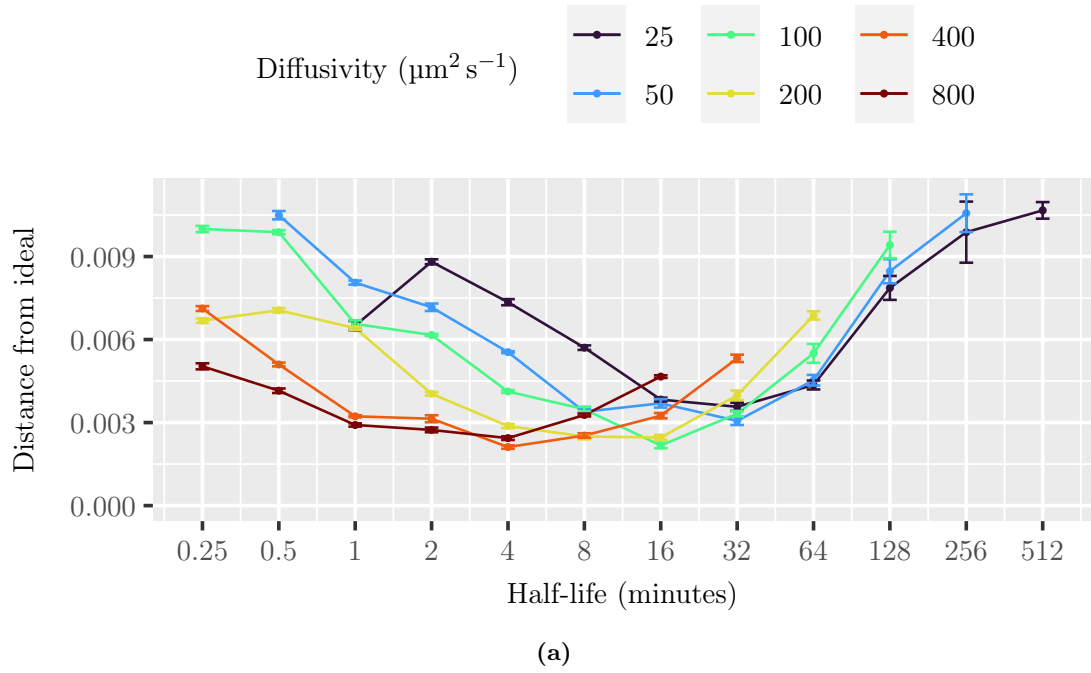


Figure 5.4: Accuracy of the heat equation test program for different morphogen physical parameters. (a) shows accuracy of the heat equation test program for different morphogen half-lives, with each curve representing a different morphogen diffusivity. (b) is a different organization of the same data as in (a); the x -axis is smoothing length, which is determined jointly by morphogen diffusivity and half-life, and each curve represents a different half-life. $N \geq 5$.

and 3.3). In particular, when agents move into once-empty locations, there is a lag before the physical morphogen field fully reflects the resulting density change (section 3.4). For the simple diffusion Morphgen program above, this lag results in a slightly slower dispersion of agents than specified in the program. Thus, when imperfect agent behavior erroneously adds dispersion to agent motion, this added dispersion can accidentally make the resulting behavior closer to the true solution of the heat equation.

To avoid such spurious results, tests in this section are different in two ways from those in previous sections. First, I introduce a bit more complexity into the heat equation Morphgen program. Second, in measuring accuracy, I use a different point of reference. I discuss each of these changes, beginning with the new Morphgen program.

The new Morphgen program is identical to that above, with the addition of a constant component to the velocity vector. This addition assumes that tissue is able to orient to gravity (or, equivalently, to any other external directional cue, such as a light source; directional cues are discussed in section 3.1.3). In the following code, $\mathbf{g}$ represents a unit vector in the direction of gravity, and $R_{\pi/2}$ is an orthogonal matrix to rotate $\pi/2$ radians.

substance tissue :

scalar fields :

ρ // Mass density

vector fields :

$\mathbf{V}$ // Tissue velocity

behavior :

$$\mathbf{V} = \frac{R_{\pi/2}\mathbf{g}}{2880} - 0.0001 \frac{\nabla \rho}{\rho}$$

The domain would be a square domain, 10 mm on a side, except that the left and right sides are joined to make it cylindrical. The speed $1/2880 \text{ mm s}^{-1}$ returns the tissue's center of mass to its original position at the end of eight hours. The top and bottom edges are given the usual homogeneous Neumann boundary conditions, so that morphogen diffusion is reflected at these walls. Agents are placed uniformly at random in a central disc of diameter 2 mm as before, but instead of the usual random orientations, agents are given an initial orientation of $R_{\pi/s}\mathbf{g}$. This alignment avoids an initial period of confused motion as agents turn toward this direction.

These modifications to the simple heat equation Morphogen program allow noise-like parameters to have a more noticeable effect by, loosely speaking, defining more moving parts for noisiness to break. I do not believe that there is any limit to how brittle a Morphogen program can be; I expect one could write a program that would perform well in an idealized MSF-SPH system but fall apart with the slightest nod toward realism. Therefore, with my choice of test program, I am trying to strike a subjective balance between behavior simple enough to suggest generalizability but complex enough to reveal some limitations of the MSF-SPH framework.

The second step I take to address the challenge of testing noise-related parameters is to define accuracy not with respect to FDM results as before, but rather with respect to MSF-SPH results under more ideal conditions. I take this step because, even with this more complex Morphogen program, simulations with small amounts of noise can give the same illusion of higher accuracy described previously when comparing to FDM results. To address this problem, I compare MSF-SPH results not with results from an FDM simulation, but with results from MSF-SPH simulations without noise or lateral motion. I perform 20 simulations under these optimal conditions, with different initial agent distributions, making for each a kernel density estimate as described in section 5.1. I then average the resulting density fields across all replicates, and use this field in place of an FDM simulation.

Default parameters for these simulations (those with lateral motion as well as those used for comparisons) are mostly the same as in table 5.1. As before, parameters are set so that noise is not modeled (excluding, of course, whatever parameter is being tested in the following sections). These settings include turning off the modeling of discrete morphogen binding events at sensors, using an exact continuous model instead. However, unlike before, sensors are modeled not only as making exact measurements, but as making instantaneous measurements as well. This assumption is unphysical, but it allows for a meaningful comparison when testing different half-lives for morphogen staying bound to sensors. Figure 5.5 shows an example simulation with the default parameters described here. A video (difftransdemo.mp4) is attached, in which one second of video represents one minute of simulation time.

5.4.2 Sensors

Sensor bias is modeled by a multiplier on each sensor’s readings, drawn from a lognormal distribution as described in section 4.6. The spread parameter for a lognormal distribution is the standard deviation of its logarithm. Figure 5.6 shows accuracy for different values of these parameters.

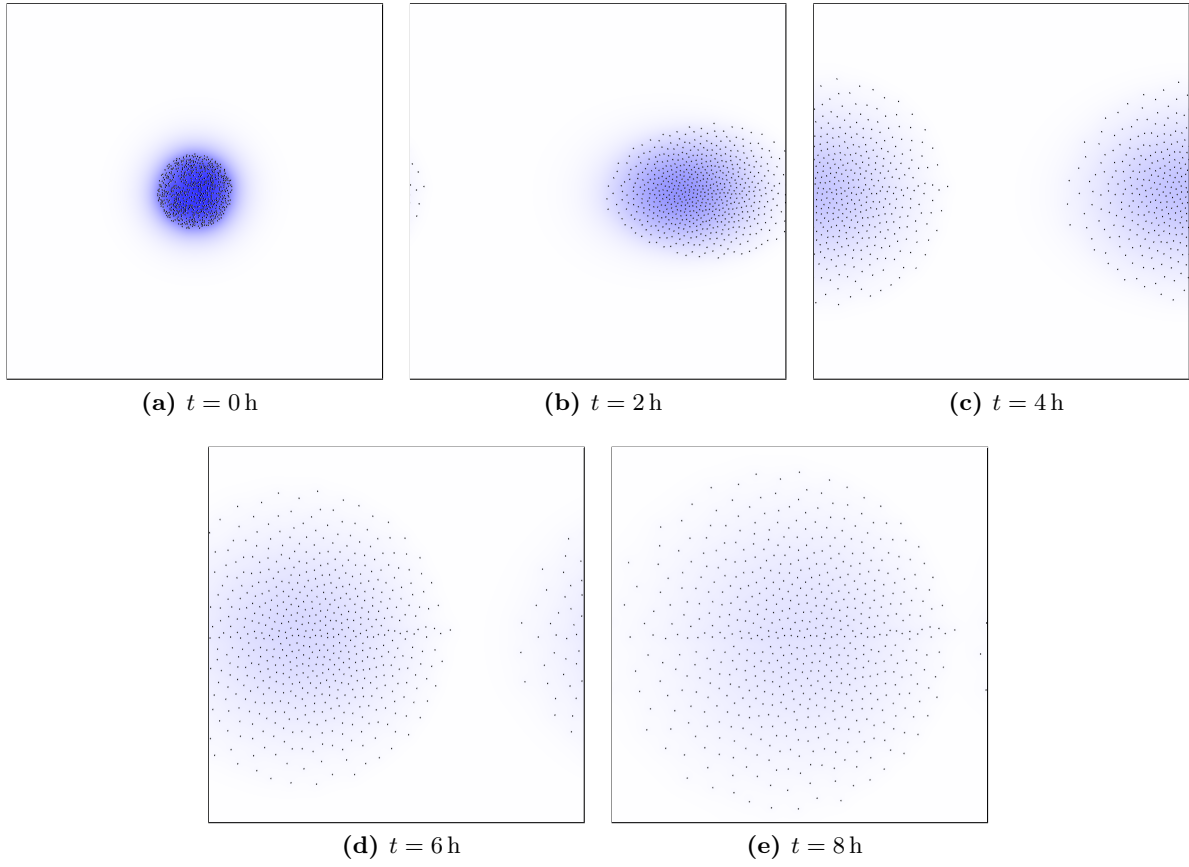


Figure 5.5: Demonstrating heat equation Morphogen program with lateral motion. Parameters are as given in table 5.1, except that sensor readings are instantaneous. Color is as in fig. 5.1.

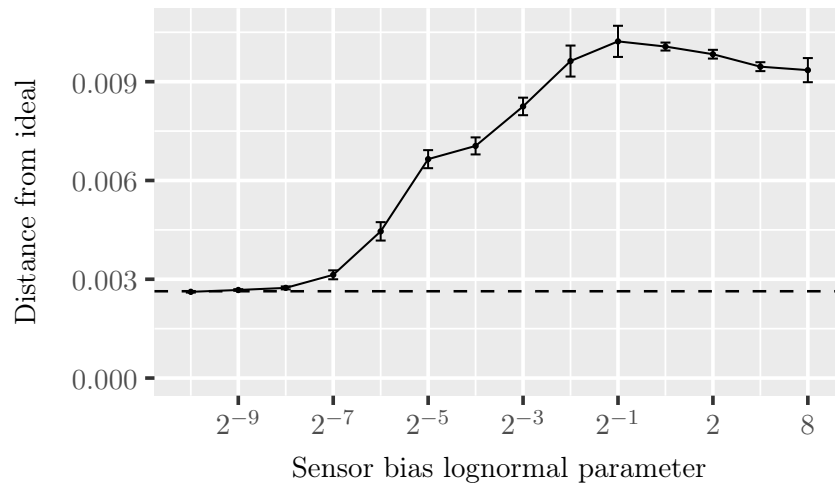


Figure 5.6: Accuracy of heat equation test program for different sensor bias spread parameters. Dashed line represents results with no bias. $N \geq 5$.

From videos of simulations at the highest levels of bias, the slight increase in accuracy from 0.25 to eight does not appear meaningful in terms of qualitative similarity to intended behavior. Instead, it seems related to interesting, curling motion of some agents at these high levels of bias. To an extent, different bias in the left versus right sensors seems to effectively hardwire a spurious apparent gradient relative to agent heading, causing an agent to turn continuously like a cat chasing their tail. Meanwhile, an agent’s estimate of an external directional cue such as gravity is not based on multiple concentration readings, so it is not subject to my bias model. The net result is that when agents are oriented toward $R_{\pi/2}\mathbf{g}$, they move quickly, but when oriented away, they move slowly or not at all, causing a curling or waving motion. It is not clear how these dynamics relate to quantitative similarity to idealized results, but they are complex enough to preclude any simple interpretation of the apparent increase in accuracy shown in fig. 5.6 for high levels of sensor bias.

Sensor noise results in part from unavoidable randomness in the frequency of binding events at the molecular level. The level of noise from this source (and it is the only source I model) therefore depends on the association rate for the sensors’ binding of morphogen molecules (section 4.6). Figure 5.7 shows that accuracy increases monotonically with association rate. These results are unsurprising; as I model sensors, higher association rates simply provide more information to sensors, with large numbers of bound morphogen molecules swamping randomness in individual binding events. However, these results come with caveats because of my simple sensor model. First, I do not model a maximum number of morphogen receptors; in a real agent, randomness in individual binding events would also affect accuracy when receptors are nearly saturated. Second, I assume that local concentration of free morphogen is not affected by the amount of morphogen bound to sensors. This assumption will be violated for sufficiently high association rates, although it may be possible to calibrate sensors to compensate for this effect. Future work could examine these effects with a more realistic sensor model.

The second parameter related to the binding of morphogen to sensors is the half-life of how long morphogen remains bound. The effect of this parameter is harder to characterize than that of association rate. Longer half-lives reduce noise, which is why I include this parameter in the present section. However, longer half-lives also mean that estimates are made using a longer exponential window, which decreases accuracy by adding lag to the system. For the heat equation test program, accuracy for different half lives are shown in fig. 5.8. Different colored lines represent different physical morphogen parameters. Though these lines are labeled according to the half-lives of free morphogen, diffusivity is also co-varied to hold smoothing length at a constant of roughly $372\text{ }\mu\text{m}$.

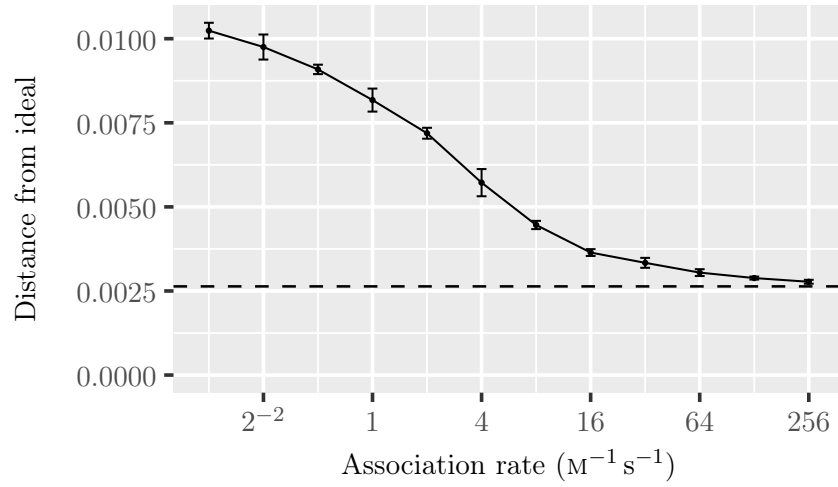


Figure 5.7: Accuracy of heat equation test program for different association constants for binding of morphogen to sensors. Dashed line represents results with exact concentration measurements. $N \geq 5$.

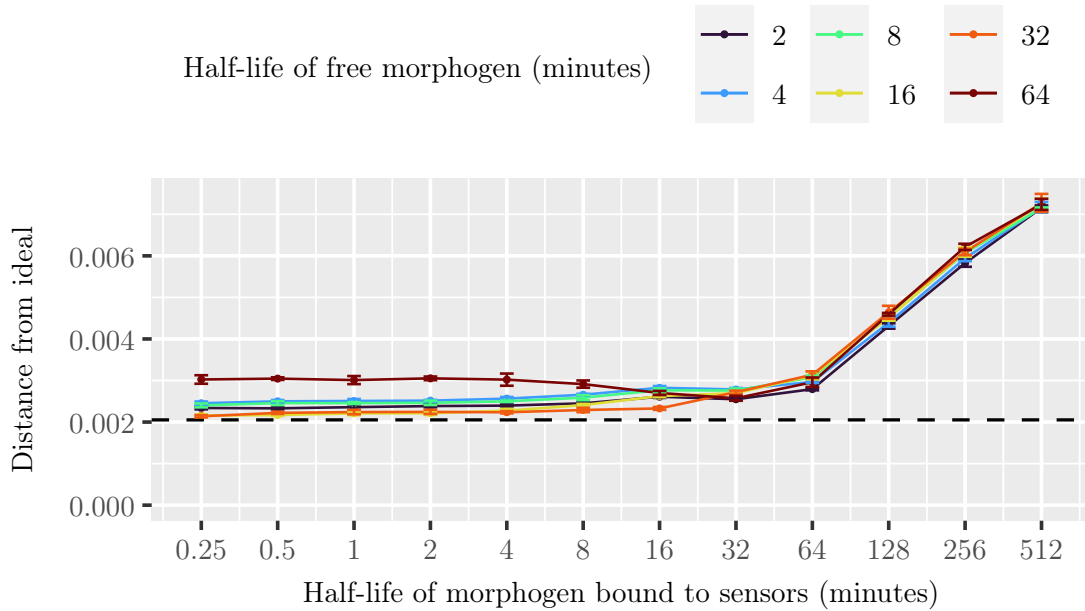


Figure 5.8: Accuracy of heat equation test program by half-life of morphogen bound to sensors, for different free morphogen half-lives. Diffusivity is co-varied to hold smoothing length at a constant of about $372 \mu\text{m}$. Dashed line shows accuracy when sensor readings are instantaneous. $N \geq 30$.

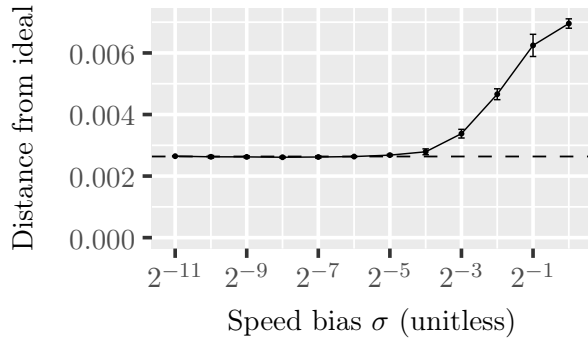
Shorter association half-lives are generally better, as expected. I tested different sensor association half-lives with various free morphogen half-lives because I expected some clear relationship between the two, but the relationship between morphogen half-life and accuracy appears more or less independent of free morphogen half-life. In retrospect, I realize that if an agent travels relatively quickly, more responsive sensors allow it to sense its changing environment even when morphogen field dynamics are slow. I cannot yet explain why association half-lives around 32 min provide the best accuracy when morphogen free half-life is 64 min.

To examine the effect of sensor association time in the abstract, these tests use an exact, continuous model for sensing. But for a given discrete association rate, very short association half-lives would hurt precision, because fewer morphogen molecules would be bound at a given time. Therefore I expect intermediate half-lives to provide the best accuracy in that situation, but more testing is needed to confirm this hypothesis.

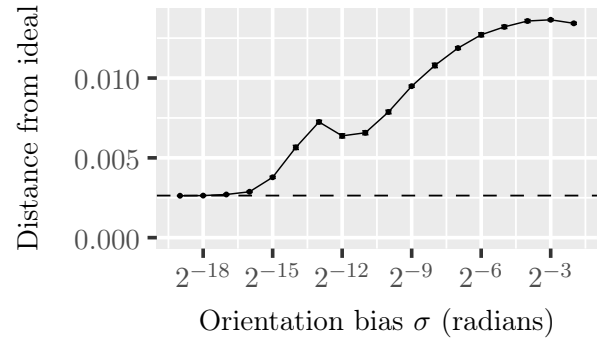
5.4.3 Motion

Imperfections in motion are modeled by two types of bias and one type of noise, as described in section 4.7. Like sensor bias, bias relating to linear speed is a multiplier drawn from a lognormal distribution. Bias relating to angular velocity is additive and so drawn from a normal distribution. The effect of each of these biases, for different spread parameters, is shown in fig. 5.9. As expected, accuracy generally gets worse with increasing spread. Figure 5.9b shows a reversal in accuracy as the standard deviation for angular velocity bias changes from 2^{-13} to 2^{-12} . Based on videos of simulations with these parameters, the apparent reason for this reversal would probably not generalize beyond the test program chosen. With parameter 2^{-13} , motion is already disordered enough that agents are distributed more or less randomly throughout the domain by the end of the simulation. With parameter 2^{-12} , the same is largely true, except that a sizeable group of agents have such large angular velocity biases that they are always trying to turn, never moving forward. These agents remain in the center of the domain, so the final mass density field is slightly more similar to ideal results (in which density remains highest near the center) than with parameter 2^{-13} .

Noise in agent motion is modeled as a random walk, with offsets in the x - and y -directions drawn from the same normal distribution at each timestep and scaled by the agent's intended speed. It is especially difficult to observe an appreciable impact of random walk noise on accuracy of a heat equation Morphogen program. This difficulty is because distributions of particles executing random walks are described by the Fokker-Planck equation, which (for constant diffusivity) is identical in



(a)



(b)

Figure 5.9: Accuracy of heat equation test program for different amounts of bias in agents' (a) linear speed and (b) angular velocity. Parameter σ is the standard deviation of the logarithm in (a) and standard deviation in (b). In each case, the dashed line indicates accuracy with no bias. $N \geq 5$.

form to the heat equation. To avoid this difficulty, I use a different approach from that described in section 5.4.1 to measure the effect on accuracy of this particular type of noise. Specifically, I use a more complex Morphogen program inspired by plant growth. I defer a detailed description of this program until section 5.8, in which this program is used more extensively. Parameters are the same as the default parameters given in that section. As shown in fig. 5.10, accuracy decreases as expected with increasing random walk noise, though accuracy is surprisingly robust over a range of smaller amounts of noise.

5.4.4 Hardware Failure

As described in section 4.9, I model hardware failure in each of an agent’s four sensors, its propulsion system, and its secretion system. Upon creation, a new agent receives a lifespan for each of these components drawn from exponential distributions. Figure 5.11 shows the effect of each of the three types of hardware failure on accuracy, for different mean times to failure (MTTF). (Each of an agent’s four sensors are modeled as failing individually, but the secretion system is modeled as failing together, even though an agent also has four secretion devices.)

Results are mostly as expected, with shorter MTTFs leading to decreased accuracy. The exception is for sensors (fig. 5.11a). Based on videos of simulations with these parameters, the decrease in accuracy from very short to moderate MTTFs for the sensor system is not particularly meaningful. In my model, I have agents continue to use the most recent valid sensor reading when a sensor has failed. At intermediate MTTF, an agent’s different sensors are likely to fail farther apart in time than for the shortest MTTFs, and therefore their final valid readings may be taken in locations more distant from each other, with a larger difference in morphogen concentration. As a result, agents may be stuck with large gradient readings that may cause ongoing extreme motion such as turning in place or traveling quickly forward. At the shortest MTTFs, different sensors are likely to fail closer together in time and thus at closer locations with more similar morphogen concentrations, and so they become stuck with gentler gradient readings. I expect a different modeling decision for how agents react to failed sensors would lead to different behavior at these short MTTFs.

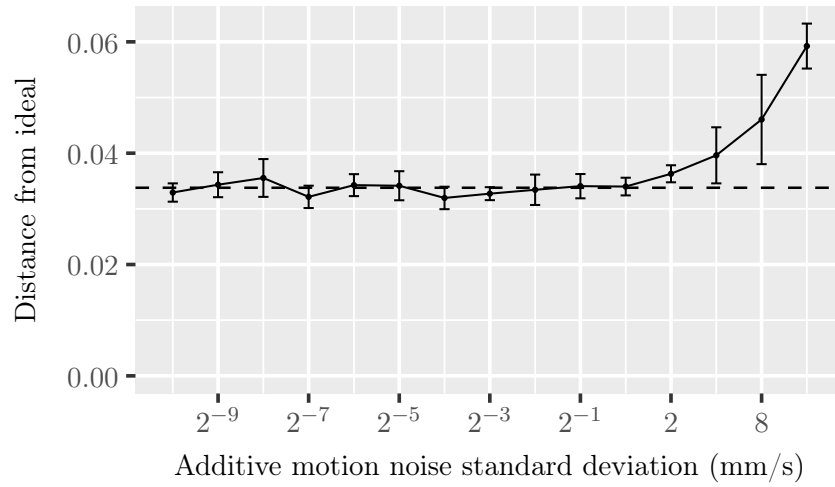


Figure 5.10: Accuracy of plant growth test program for different standard deviations for random walk noise. Dashed line represents results with no noise. $N \geq 7$.

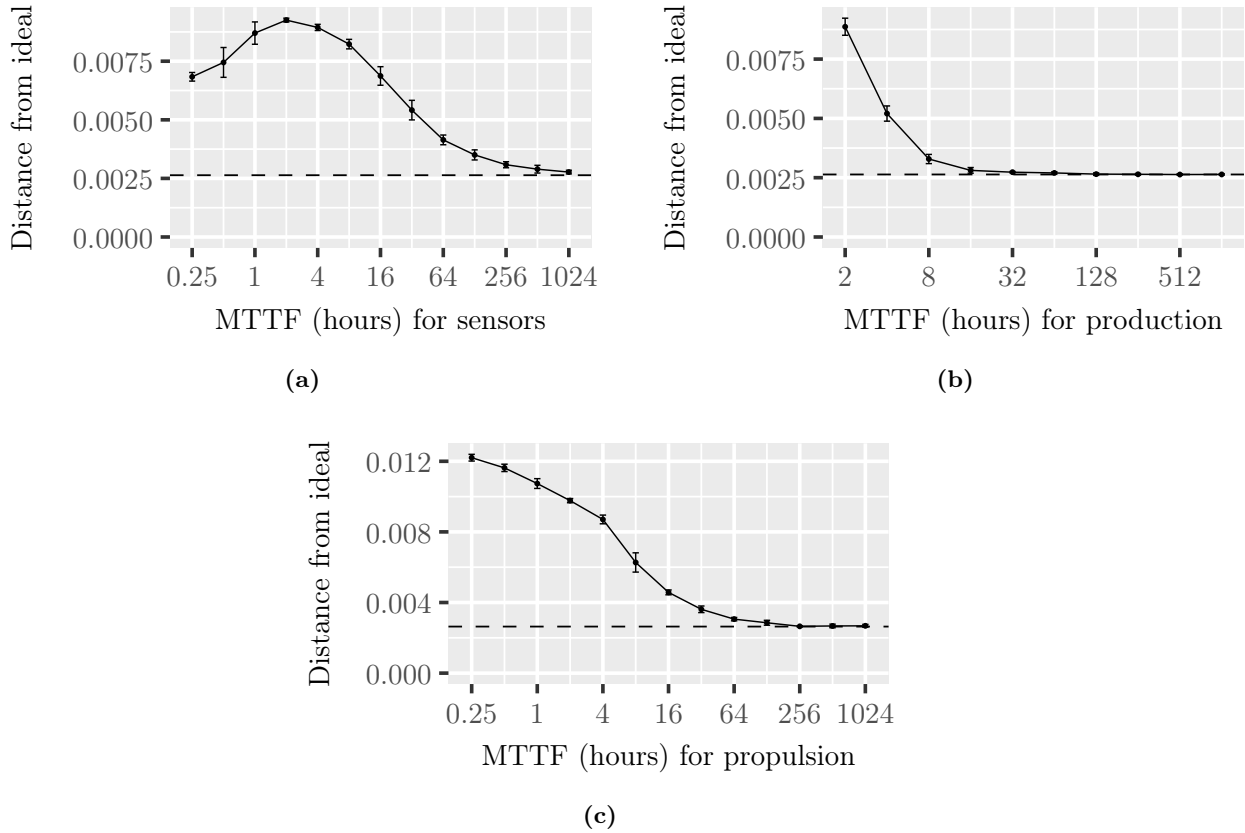


Figure 5.11: Accuracy of heat equation test program for mean times to failure of (a) sensors, (b) production devices, and (c) propulsion. Sensors are treated individually, whereas production devices are treated as one system. $N \geq 5$.

5.5 Agent Speed and Calibration

As agents' speeds increase relative to morphogen diffusion rate, the more is violated the idealized assumption from section 3.3.1 that agent speed will be negligible relative to diffusivity. As this assumption is violated, agents' morphogen fields become distorted, and neither morphogen concentration nor its gradient will be consistent with the idealized MSFs presented in that section. To some extent, an agent can use calibration to account for distortions due to its own field, as discussed in sections 3.4.2 and 4.1. However, agents cannot account for their neighbors' distorted fields, because they do not know the positions or velocities of their individual neighbors.

In this section I test the ability of calibration to address these challenges. I also look at the effect on accuracy of speed relative to diffusivity. The Morphgen program from the previous section, in which the heat equation is combined with constant lateral motion, is also well-suited for examining the effect of agent speed on accuracy. In this section I focus on the relationship between speed and distorted morphogen fields, so I assume sensor dynamics are instantaneous, as I did in section 5.4.1 for comparison purposes. Although unrealistic, this assumption allows us to separate the effect of distorted smoothing functions from the effect of sensor dynamics.

As a variable to help isolate the effect of speed on accuracy, I introduce an artificial speed multiplier. The velocity field determined by the Morphgen program is multiplied by this factor, and simulation time is divided by the same factor. Ideal final results for comparison are thus unchanged. Once again, to address the issue involving illusory gains in accuracy with additional dispersive error, results are compared to MSF-SPH results under optimal conditions, and without lateral motion, as in the previous section.

Figure 5.12 shows the relationship between agent speed and diffusivity. The different lines in the figure are labeled according to physical morphogen diffusivity, but morphogen half-life is co-varied to hold smoothing length constant at roughly $372\text{ }\mu\text{m}$. (This is the smoothing length implied by our usual defaults of $100\text{ }\mu\text{m}^2\text{ s}^{-1}$ diffusivity and 16 min half-life.) The x -axis is the artificial speed multiplier divided by diffusivity. This choice of axis causes the curves for different diffusivities to align closely, which illustrates that the relevant influence on accuracy is, as expected, agent speed relative to diffusivity. That accuracy still suffers at high relative speed is also expected; calibration can partially compensate for the distortion in an agent's own morphogen field, but not in those of its neighbors.

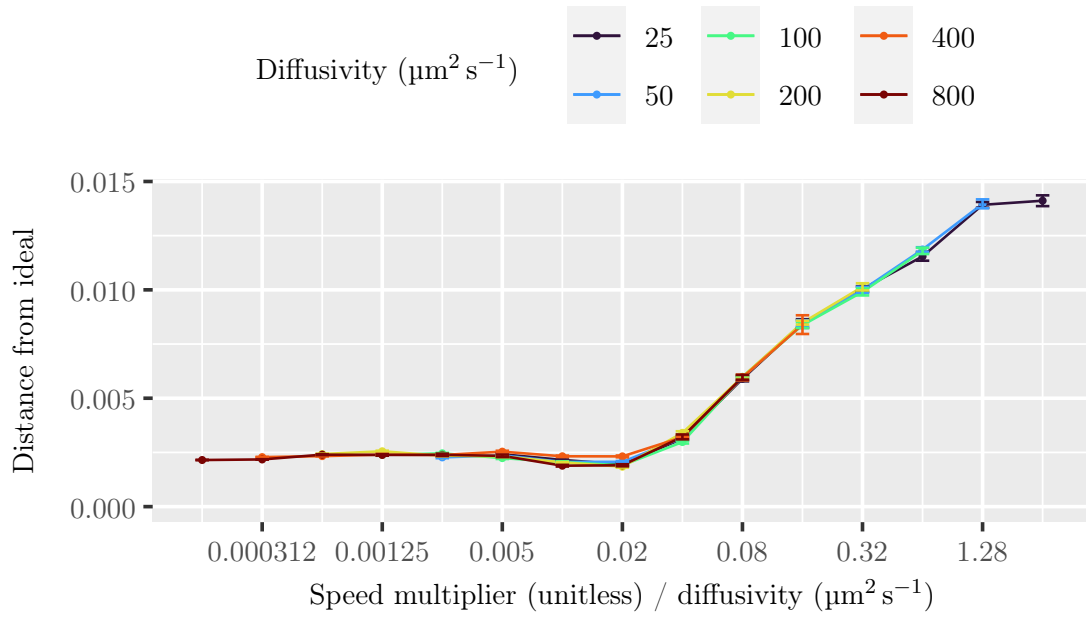


Figure 5.12: Accuracy for heat equation test program by ratio of artificial speed multiplier to diffusivity. Each colored line represents a different morphogen diffusivity. $N \geq 5$.

To explore the extent to which calibration mitigates smoothing function distortion at speed, I perform similar tests with no calibration, as well as with calibration for morphogen concentration but not gradient and vice-versa. For these tests I fix diffusivity at $100\mu\text{m}^2\text{s}^{-1}$, as usual. Results are shown in fig. 5.13.

As expected, calibration improves accuracy at most speeds. At sufficiently slow speeds, smoothing function distortion is negligible, so calibration would not be expected to help below some minimal speed. Calibration does not appear to help much at higher speeds either; in fact, calibration for concentration at high speeds appears detrimental. However, these appearances may be the result of a weakness in my testing method, which I now explain.

As usual, I allow mass density morphogen to build up before starting the simulation, as though agents had been activated but not moving for a while. This allows a morphogen field to build up within the initial disc. But at very high lateral speeds, this morphogen field still retains much of its original shape by the time agents circumnavigate the cylindrical domain and return to this location. This interaction causes various unintended dynamics and renders results for very high speeds difficult to interpret.

It remains the case that even at intermediate speeds, calibration for concentration does not appear to improve accuracy. This result may relate to the different speeds at which distortion of gradient and distortion of shape are highest. As speed increases from zero, more morphogen is found behind an agent than in front, creating a spurious gradient, but at first the average concentration around the agent is changed little. At these speeds, calibration for gradient is most needed. At higher speeds, the agent outruns most of its own morphogen field; at this point, concentration around the agent becomes much lower, but the spurious gradient is reduced because there is simply not enough morphogen around the agent to form a steep gradient. At these speeds, calibration for concentration is most needed. To confirm this idea, as well as to obtain more meaningful results generally for very high speeds, more work is needed to find a suitable test. It may prove sufficient to retain the same Morphogen test program used here (a heat equation with added lateral motion), but using a cylindrical world perhaps three or four times wider. Results could then be measured before agents return to their initial locations.

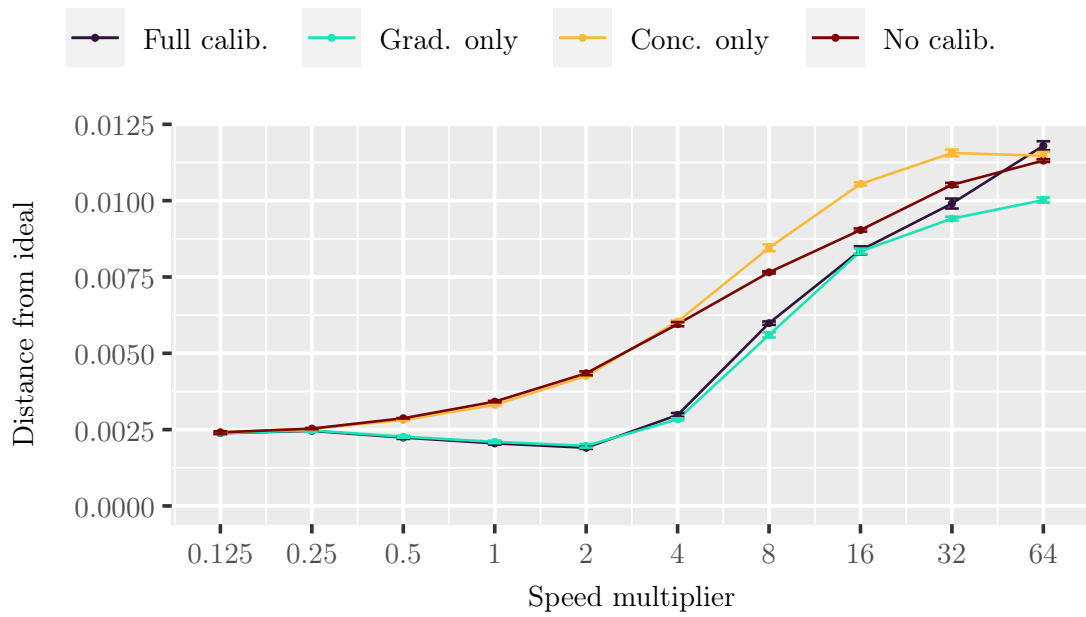


Figure 5.13: Accuracy for heat equation test program by artificial speed multiplier. Each colored line reflects which kind of calibration data were used. $N \geq 23$.

5.6 Correction for Laplacian Operator

Section 3.4.1 describes a problem for MSF-SPH associated with changes in field values, along with an approach to address it. When a given agent's field value changes, it changes its secretion rate, but it takes time for its morphogen field to reach a new steady state following this change. This creates a lag in agents' mutual communication of changes in field values. This lag is especially detrimental to Laplacian estimates, which an agent makes by comparing its internal value for a given field with its MSF-SPH estimate of the field's value.

Here I test my approach to mitigating this problem. This approach involves agents assuming, for purposes of the Laplacian operator, that their neighbors' values for a given field are changing at the same rate as their own. I use the following program to test the accuracy of the Laplacian operator. For simplicity, I assume agents have the ability to sense the presence of a cue B found in a disc-shaped region.

```

cues :

       $B$  // Arbitrary region

substance tissue :

      scalar fields :

       $A$  // Field to be made paraboloid-shaped over  $B$ 

      behavior :

       $\nabla A = 0.001[B](\nabla^2 A + 1.0)$ 
```

This program, initialized with field A at zero, evolves toward field A having the shape of a circular paraboloid whose Laplacian is -1.0 everywhere. The domain is, as before, a square 10 mm on a side, and the disc with cue B is in the center of the domain and has a 6 mm diameter. Agents are placed uniformly at random throughout the domain to a relative density of 0.1 (or 7958 agents), and the program is simulated for one hour. Other parameters are the same as those given in table 5.1. Although Morphogen programs need not simulate any physical scenario, this program happens to correspond to a uniformly heated disc in a bath of constant temperature.

Figure 5.14 shows the comparison between agent-based simulations, with and without the Laplacian correction from section 3.4.1. Each is compared with ideal results. As in the above sections, these ideal results are found using an FDM simulation of the same program. Because the

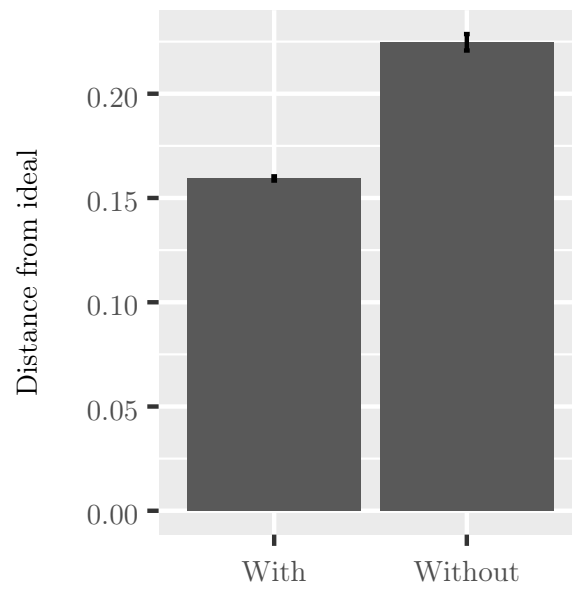


Figure 5.14: Comparing paraboloid generation to ideal results for heated disk in bath, with and without correction for Laplacian operator. $N = 5$.

field we are comparing with FDM results is an abstract field rather than a mass density field, I use a more straightforward method of comparison: I simply take the average of absolute differences between each agent's value for A and the corresponding point from the FDM field. This has the effect of weighting the comparison according to where agents are located, which seems appropriate because the pattern the program generates really exists only at the agents' locations. As expected, the Morphgen program is more accurate with a corrected Laplacian operator.

5.7 Upwind Scheme

Section 3.9 introduces an upwind scheme for advection that can be used with MSF-SPH. I test it with the following program that maintains an initial pattern in a field A against agent motion. To maintain the simplicity of all agents traveling the same direction, this program assumes that agents are able to orient to gravity (or, equivalently, to any other external directional cue, such as a light source).

```

substance tissue :

    scalar fields :

         $A$     // Pattern to be advected

    vector fields :

         $\mathbf{V}$     // Tissue velocity

    behavior :

         $\nabla A = \text{VELDOTGRADUPSTREAM}(A, \mathbf{V})$ 
            + 0.1[ $A > 1.0$ ](1.0 -  $A$ )

         $\mathbf{V} = 0.001 R_{\pi/2} \mathbf{g}$ 

```

Here $\mathbf{g}$ represents the direction of gravity and $R_{\pi/2}$ is an orthogonal matrix to rotate $\pi/2$ radians.

The final line of this program limits the value of A to a maximum of one, which would ideally be the maximum value of field A based on the initial pattern given above. This limiting is helpful because NSF-SPH cannot communicate values below zero without an additional channel (section 3.1.3), which effectively limits field A to be nonnegative. This effective limitation serves as a ratchet mechanism: inaccuracies can bring field levels above one but not below zero, leading

to ever-higher field values. Limiting values to one does not correct for these inaccuracies, but it mitigates their damage to the pattern. (Another approach would be to use an additional morphogen to communicate negative values, but I do not pursue this idea, because my focus in this section is to test the upwinding scheme.)

The initial pattern is the two-dimensional Gaussian function $A = \exp(-(r/3)^2)$ where r is distance from the center of the domain. The domain would be a square domain, 10 mm on a side, except that the left and right sides connect to make it cylindrical. Thus the agents travel continually around the cylinder, passing through the pattern repeatedly. The top and bottom edges are given the usual homogeneous Neumann boundary conditions, reflecting morphogen diffusion. As in section 5.6, agents are placed uniformly at random throughout the domain to a relative density of 0.1 (7958 agents). But instead of the usual random orientations, agents are initially oriented in their direction of travel, to avoid the initial period of confused motion that would otherwise result as they turned. Simulation time is one hour, and other parameters are as given in table 5.1. For illustration, initial and final fields are shown in fig. 5.15, and a video (advectdemo.mp4) is attached.

Figure 5.16 compares the accuracy of this program with its equivalent without the upwind scheme—that is, with $\text{VELDOTGRADUPSTREAM}(A, \mathbf{V})$ replaced by $A \cdot \mathbf{V}$. As in section 5.6 above, results are compared to an ideal result by taking the average distance between each agent’s field value and the ideal value at that location. In this case, however, we do not need an FDM simulation, because field A is unchanging in the ideal solution. Although more modest than I expected, the upwind scheme does improve accuracy for this paraboloid-forming Morphogen program.

5.8 Demonstrating Growth

Section 3.7 presents methods for implementing density fields that change value independently from velocity divergence through reproduction (or recruitment) and apoptosis (or deactivation) of agents. I demonstrate this facility, and compare implementation methods and parameter values, with the following simple program inspired by a growing plant stem. As in section 5.7, I assume agents can sense the direction of gravity.

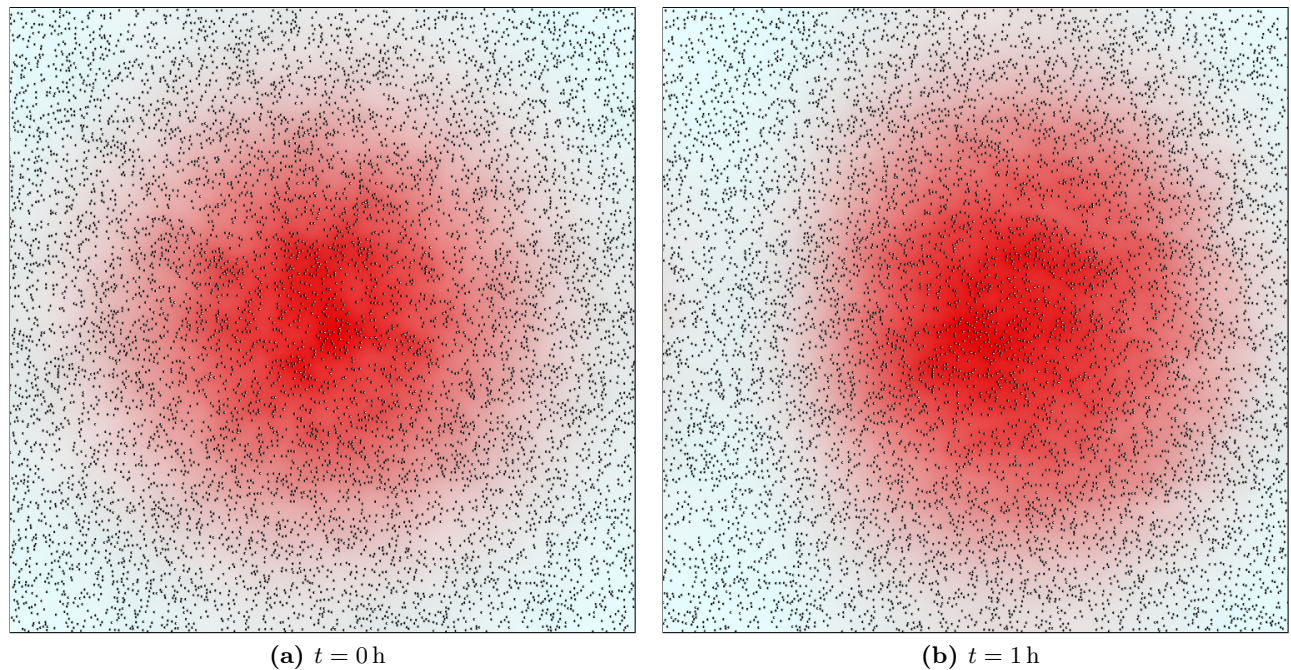


Figure 5.15: Demonstrating advection Morphgen program. Parameters are as given in table 5.1, except as described in the text. The morphogen representing the density field (which is not interesting in this instance) is shown in cyan. The morphogen representing the Gaussian pattern held in place by advection is shown in red. Each are scaled so that full saturation occurs at the concentrations representing a field value of one.

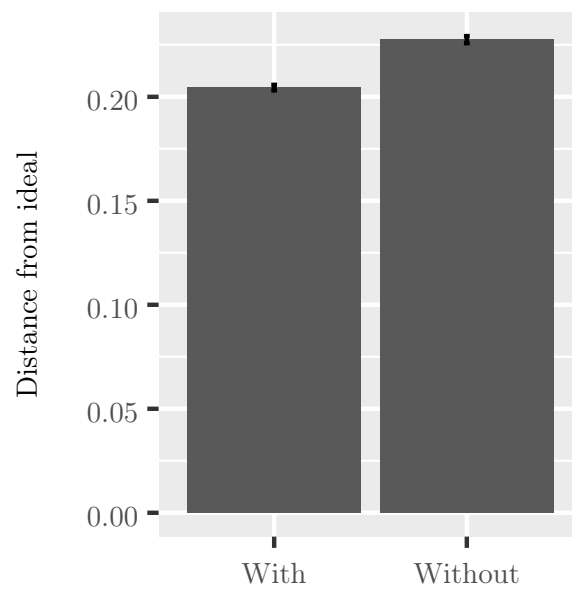


Figure 5.16: Comparing pattern advection to ideal results, with and without upwind scheme. $N = 5$.

substance plantStem :

scalar fields :

ρ // Mass density

C // Color field

T // Timer to delay growth until meristem forms

M // Meristem

vector fields :

$\mathbf{V}$ // Tissue velocity

behavior :

$C = 1$

$\text{DT} = 1$

$\Gamma = [T > 10800]1$

$\text{D}\rho = 0$

let $\beta = [\text{EDGEDETECTCOLOR}(\rho) > 0.3 - 0.1\Gamma]$

let $\gamma = \cos^{-1} \left(\frac{\mathbf{g} \cdot \nabla C}{\|\mathbf{g}\| \|\nabla C\|} \right)$

$\text{D}M = 0.1[\gamma < 0.4\pi \wedge \beta = 1](1 - M)$

$- 0.1[\gamma > 0.6\pi \wedge \beta = 1]M$

$+ 0.02[\beta = 0 \vee 0.4\pi \leq \gamma \leq 0.6\pi]\nabla^2 M$

let $\mathbf{V}' = -0.0001\Gamma M \mathbf{g} - 0.0004[\beta = 0]\nabla \rho$

let $\mathbf{V}_{\max} = 0.01$

$\mathbf{V} = \mathbf{V}_{\max} \tanh \left(\frac{\|\mathbf{V}'\|}{\mathbf{V}_{\max}} \right) \frac{\mathbf{V}'}{\|\mathbf{V}'\|}$

I initialize this program with agents placed uniformly at random within a square 3 mm on a side near the bottom of a domain 10 mm high and 6 mm wide. Tissue sets a high value for field M (for meristem) where tissue is on an edge and where the mass density gradient is similar to the direction of gravity, which indicates a position near the tissue's top edge. I refer to increases in M informally as differentiation, by analogy with plant morphogenesis, but it is not differentiation in

the more precise sense that I define in section 3.6. The differentiation described in that section is needed only when tissues of different differentiation states overlap spatially.

Color field C is used to locate the edge of the tissue, as described in section 3.8.2. To stay true to the evolutionary PDE theme of the subset of Morphogen that I choose to work with, I should have specified this as something like $\partial C += 0.1(1 - C)$. Such a change equation would clearly have the same effect in this context as $C = 1$, which is what I actually used in simulation. (This simple assignment is also consistent with how I plan to implement the color field as a compiler feature to be exposed as an edge-finding primitive.)

Field M diffuses through the tissue, creating a smooth gradation tapering off downward. This is an abstract diffusion defined by the Morphogen application, independent of the diffusion and degradation of the physical morphogen that communicates field M . As the meristem field takes shape, tissue moves down the density gradient, except on the edges. This motion serves to even out the density field, which improves the accuracy of the meristem gradation.

After three hours, the above processes continue, but in addition, the tissue begins to move opposite the direction of gravity, at a speed proportional to M . Also at the three-hour mark, the density control binary field Γ is turned on by setting it to one. At this point, with Γ set to one and $\partial\rho$ to zero, the tissue begins trying to maintain its density against the attenuating effect of its upward motion.

Results over twenty hours are shown in fig. 5.17 and in an attached video (growthdemo.mp4). The top of the stem becomes somewhat ragged as growth progresses; I believe this is largely because a slightly uneven distribution of agents along the top edge leads to random variations in the angle between gravity and the measured density gradient. I have been able to obtain smoother growth by fixing the M field after three hours; in this case the gradation in the field stretches as growth progresses, but the top edge remains relatively even. However, this alternative approach sacrifices some of the above program's robustness. So long as M remains determined dynamically by the continuing relationship between gravity, the density gradient, and the color field gradient, the stem has an ability to recover from damage. Figure 5.18 shows the stem's recovery after having a midsection destroyed at the twelve hour mark.

Section 3.7 presents two alternative approaches to satisfying $\partial\rho$, one corresponding to integral control and the other to proportional control. The results shown in figs. 5.17 and 5.18 use the proportional control approach with a leak rate of 64h, a choice justified by results presented in the next section. Other parameters not mentioned above are given in table 5.2. Except where

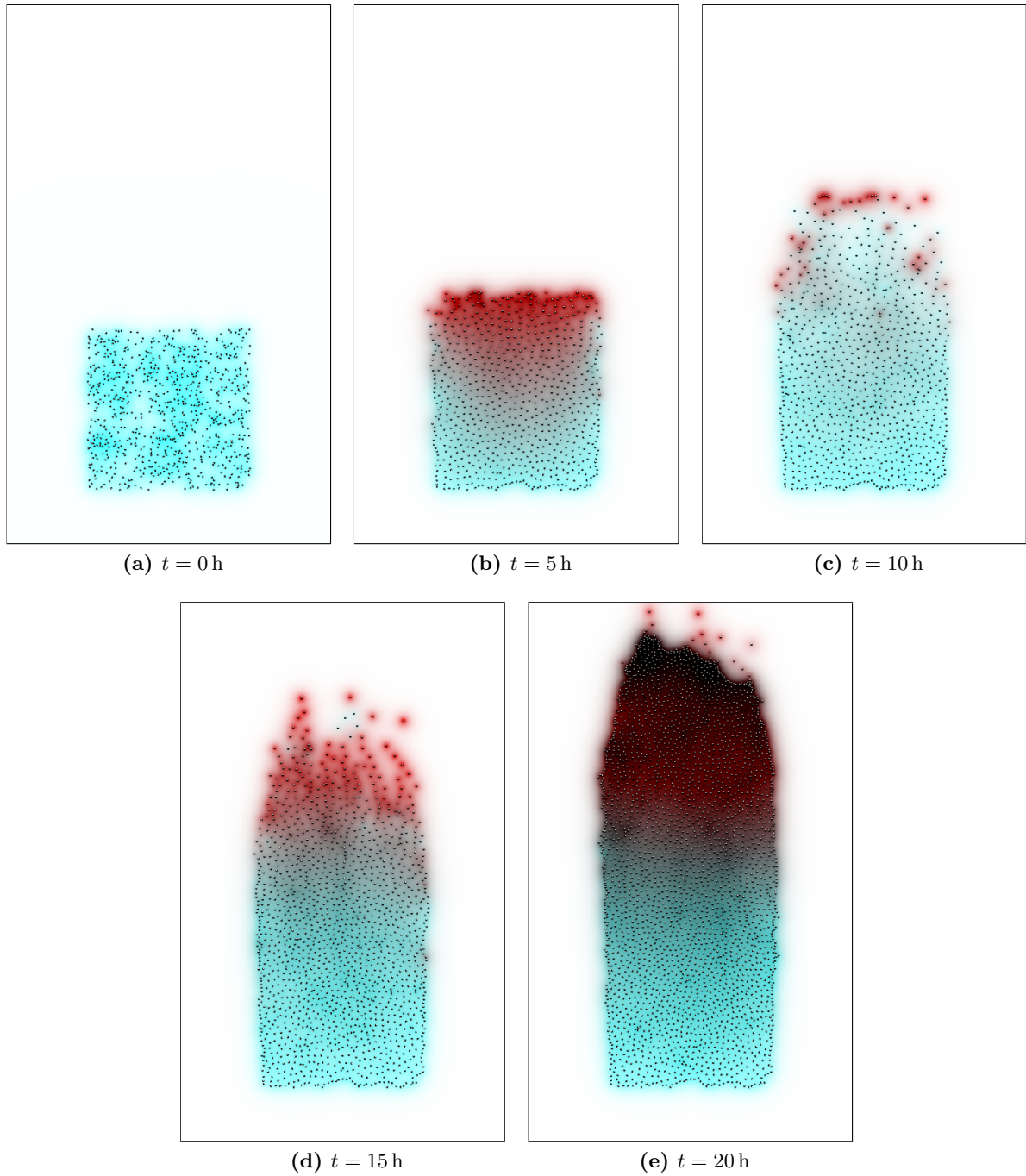


Figure 5.17: Demonstrating stem growth program. The first three hours are dedicated to smoothing the density field and establishing the meristem field. After three hours, tissue travels upward at a speed proportional to the value of the meristem field. Cyan represents the density morphogen and red the meristem morphogen. Colors are scaled so that full saturation occurs at concentrations representing field values 0.2 and 1.0, respectively. A color field used to identify edges is not shown; superficially, it appears similar to the density field.

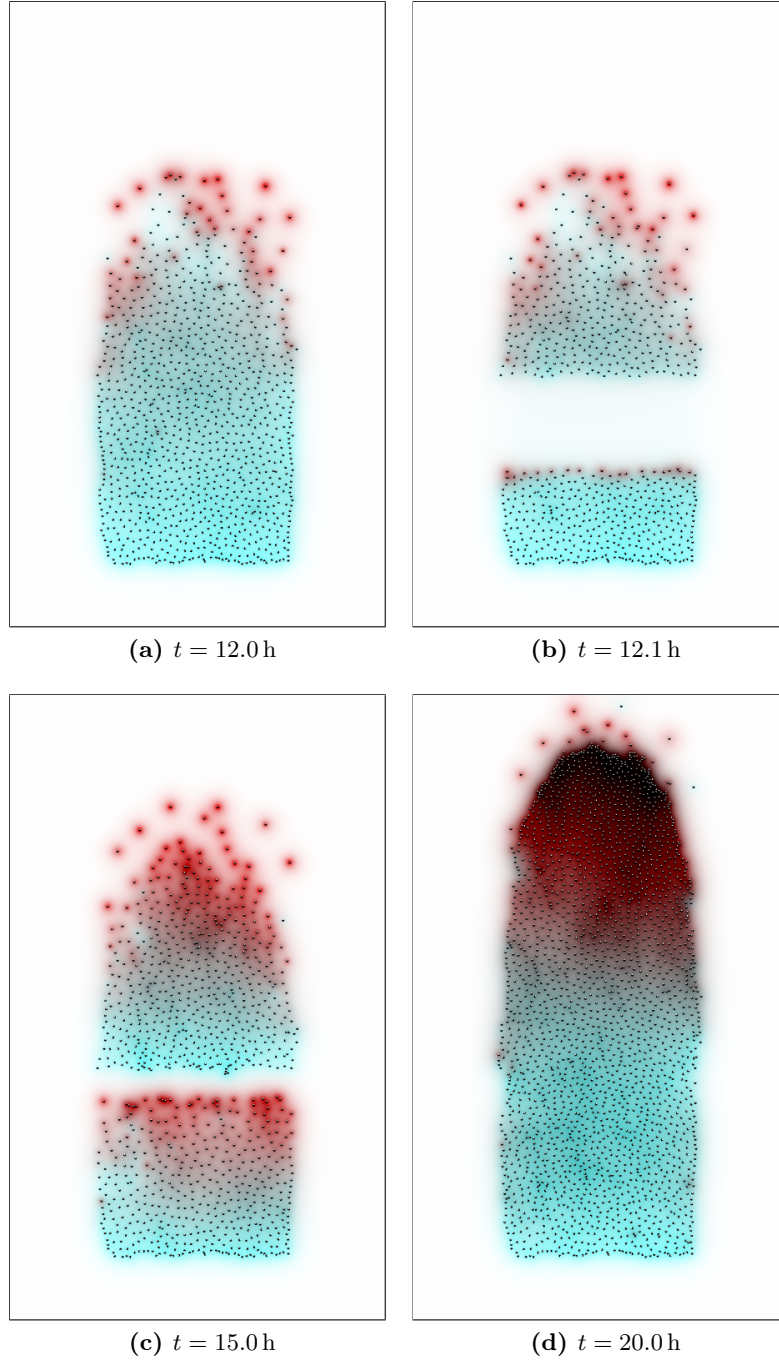


Figure 5.18: Robustness of stem growth program to cutting out agents. At $t = 12$ h, all agents whose position has a y -component between 2.5 and 4.0 are destroyed. The newly exposed upper surface re-differentiates into meristem and begins growing upward, healing the wound. Colors are as in fig. 5.17.

Table 5.2: Default parameters for growth program

Parameter	Value
Total time	18 h
Timestep	0.53 s
Pixels per millimeter in each direction	75
Agent diameter	40 μm
Agent body eccentricity	0
Initial relative density	0.1 (~ 716 agents)
Morphogen diffusivity	100 $\mu\text{m}^2 \text{s}^{-1}$
Morphogen half-life (mass density)	2 second
Morphogen half-life (meristem)	2 second
Morphogen half-life (color field)	4 second
Sensor association half-life	1 minute
Integral leak rate for proportional control	64 h

mentioned, these parameters are also used in the tests presented in the next two sections, which also use this plant stem program.

5.9 Integral and Proportional Control

Section 3.7.1 presents an approach to satisfying $\mathbb{D}\rho$ by maintaining a running integral of this quantity and comparing this target density with current measured density. Section 3.7.2 presents an alternative approach to the same end involving a leaky integral of density estimates. These two approaches correspond to integral and proportional control, as explained in section 3.7.4. Proportional control in particular introduces a new parameter to the model, the leak rate of the leaky integral. I first look at the effect of a range of values for this parameter. Having identified a reasonable value for leak rate, I then use this leak rate in comparing integral and proportional control with each other.

Each test uses the same plant stem program introduced in section 5.8 above, with the same parameters shown in table 5.2. As usual, I compare each MSF-SPH simulation with ideal results. As in section 5.7, no FDM simulation is needed for the plant growth program; ideal results are just a tall rectangular region corresponding to the initial square region, extended upward according to the speed of tissue where $M = 1$. The density of this region is just our starting density 0.1, and for our initial height of 3 mm, speed of 0.0001 mm s^{-1} , and simulation time of 18 h, the ideal final height of the rectangle is 8.4 mm. To compare these results with those from an agent-based simulation, I use the same approach as described in section 5.1. I make a grid containing the same size as the simulation grid and containing an image of the ideal final rectangle. From the final agent positions in the MSF-SPH simulation, I make a kernel density estimate at every grid cell and average the absolute differences between each pair of corresponding cells in the two grids.

In examining the effect on accuracy of different leak rates, I find it more intuitive to think in terms of half-life, in minutes, rather than directly in terms of leak rate, which in my simulator has units of inverse seconds. Denoting each quantity as $t_{1/2,l}$ and k_l respectively, the two are related by $k_l = (\ln 2)/(60t_{1/2,l})$. Accuracies for a range of half-lives are shown in fig. 5.19a.

In comparing integral and proportionate control, I use for proportional control the most accurate of the half-lives tested, 64 h, which corresponds to a leak rate of about $3 \times 10^{-6} \text{ s}^{-1}$. Results of this comparison are shown in fig. 5.20.

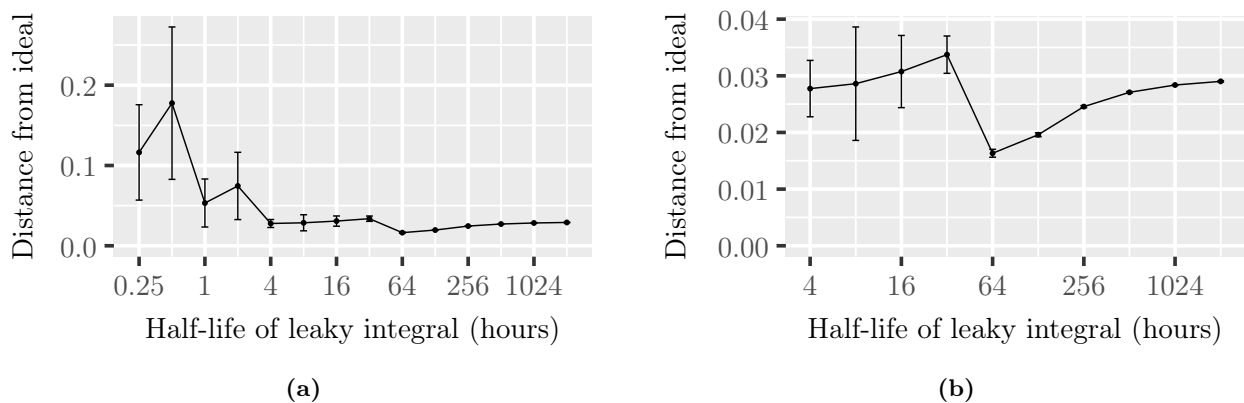


Figure 5.19: Accuracy of growth test program, using proportional control, for different half-lives for the leaky integral used in this approach. (a) shows results across the full tested range; however, results for half-lives 0.25 h and 0.5 h are biased, because some simulations with large amounts of excessive growth were unable to complete. For clarity, (b) shows the same data as (a) for half-lives 4 h and up only. $N \geq 20$.

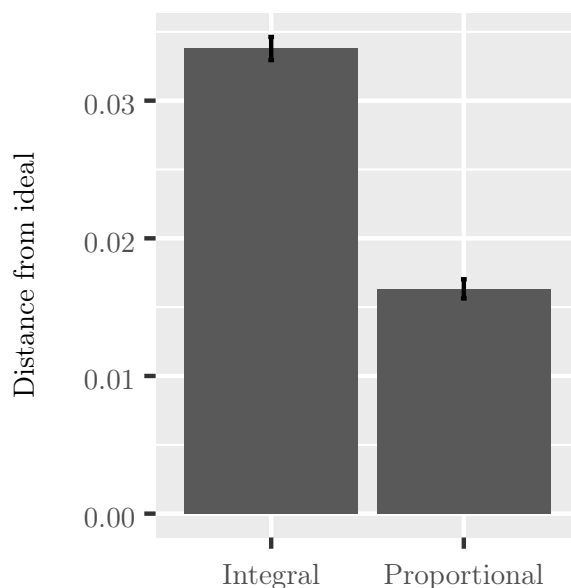


Figure 5.20: Comparing integral and proportional control approaches. $N \geq 20$.

I expected the observed trade-off between faster and slower leak rates. An overly fast leak rate would be subject to noise and provide inaccurate estimates of density change, but an overly slow leak rate would take too long to reflect density changes in its estimate. But I was surprised by the long duration of the best leaky integral half-life observed here (64 h), which is longer than the simulation time. I was also surprised by the apparent inferiority of integral control to proportional control with this leak rate, because integral control uses a more direct method of estimating density change.

Observing videos of these simulations, I believe both these surprises may relate to selection effects that cause tissue density to drift upward with time. With respect to integral control, any slightly excessive target density in some agents will be amplified by selection: such agents will reproduce (or recruit) more often than others, passing along their excessive target density to these children, which will in turn reproduce excessively. The correction described in section 3.7.5 for correcting the target density a recruited child agent inherits from its parent helps avoid this problem, but does not eliminate it. (The following section will examine the benefit of this correction.)

In contrast to integral control, in proportional control there is no target density to be passed to new agents; however, the leaky integral value is passed to new agents. This inheritance may cause a similar problem as in integral control, and may be amenable to an analogous correction approach in the future. In fact, because my proportional control approach currently lacks such a correction, the apparent superiority of proportional to integral control may be illusory. Because the proportional control approach relies on a leaky integral to estimate density change rate, this estimate will inevitably lag behind actual density changes, with the timescale of lag determined by the leak rate chosen. Therefore, the apparent good performance of the surprisingly long 64 h leaky integral half-life may be an artifact, with inevitable overgrowth simply delayed and offset by a long initial lag period.

More investigation is needed to test the above speculation. An additional, potentially supporting observation is the high spread of results seen at shorter leak rates in fig. 5.19a. On examination of simulation videos, this high spread seems caused by subsets of trials in which a more extreme version of cancer-like overgrowth occurs. An example simulation, for which the leaky integral half-life is 15 h, is shown in fig. 5.21 and in an attached video (overgrowth.mp4). Very rapid growth begins at a particular location, typically near the edge of tissue, and continues until the entire domain is filled with agents. At intermediate leak rates, this overgrowth occurs only occasionally, but at the fastest

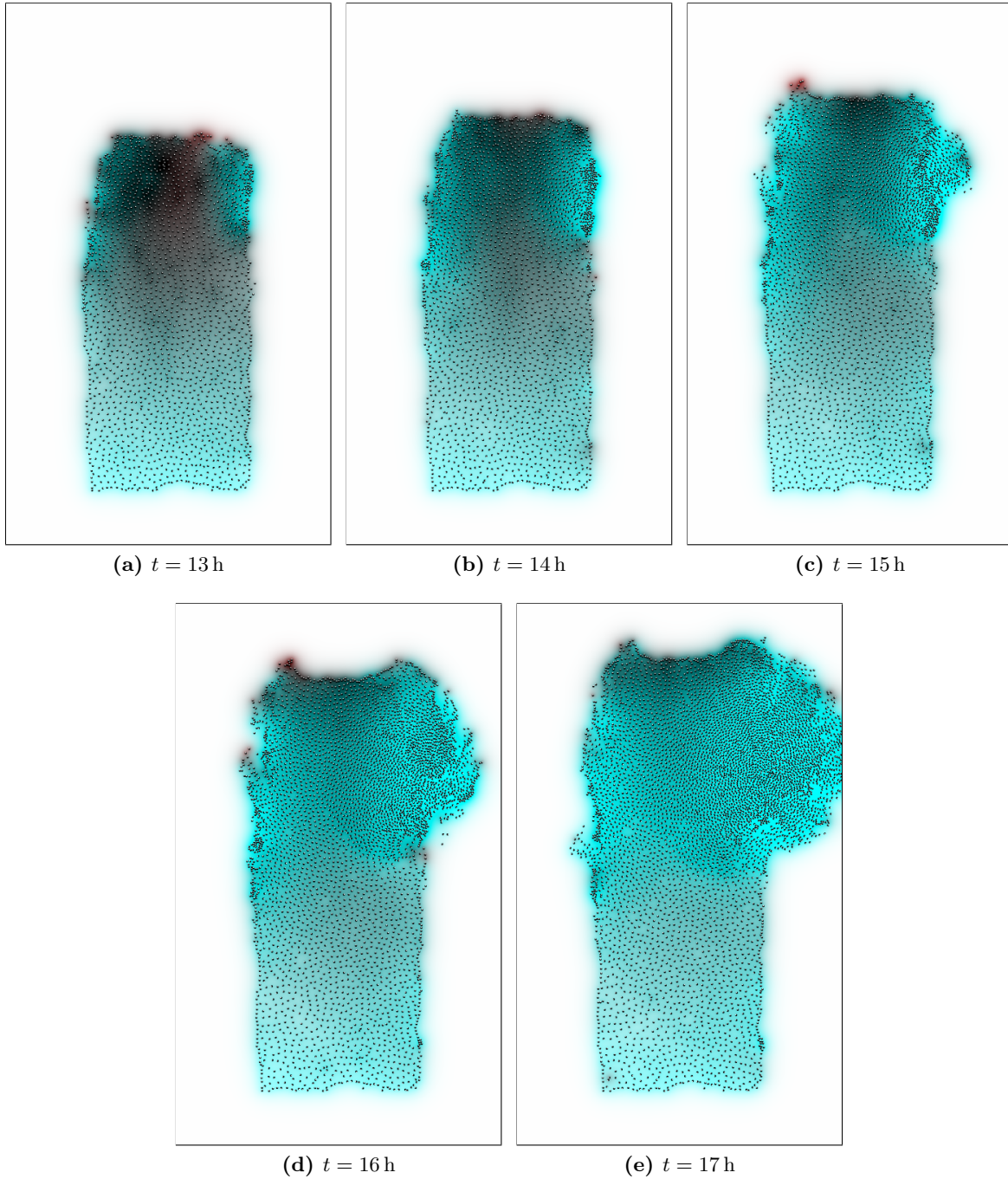


Figure 5.21: Example of tumor-like overgrowth in plant stem growth program. Leaky integral half-life is 15 h in this simulation. For the eighteen-hour simulations tested, this type of overgrowth is only occasional at a relatively slow leak rate such as this, but occurs reliably with faster leak rates.

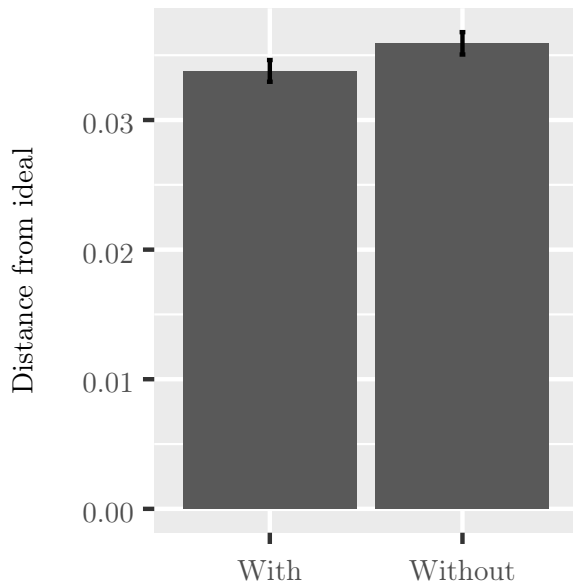
leak rates it occurs reliably. This problem resembles that which led me to introduce the correction for target density in my integral control approach (section 3.7.5).

5.10 Density Corrections for Recruitment

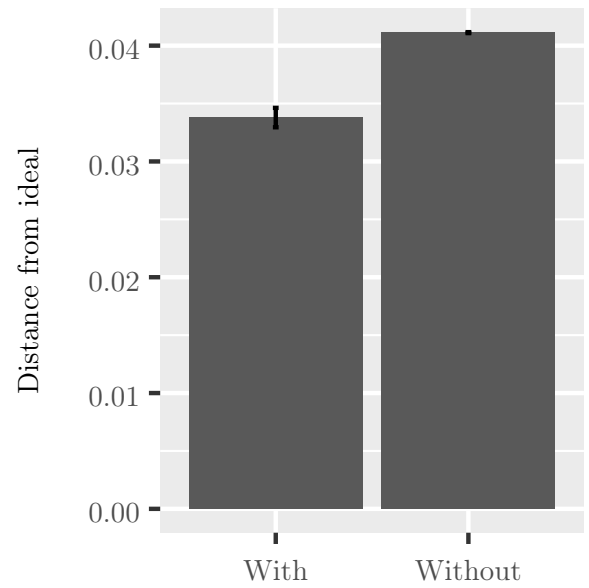
My simplistic reproduction model, in which new agents are simply created near a parent when the latter chooses to reproduce, can be seen as a crude model of recruitment of new agents from an inactive pool. This idea is discussed in section 3.7.5, which also addresses two challenges raised by recruitment. First, new agents must adjust their MSF-SPH estimates during the period before their own morphogen fields have had time to build up to steady state. Second, when the integral control approach is used, new agents must adjust the target densities inherited from their parents to account for their slightly different position in the tissue. Using the same plant growth program as before, I evaluate the effect of each of these strategies by comparing results with and without each strategy. Integral control is used in both cases; I expect the pre-steady-state issue to affect performance similarly whether integral or proportional control is used, and the issue of target densities only arises in the context of integral control.

Figure 5.22a shows the effect on accuracy of correcting new agents' MSF-SPH estimates. The benefit of this correction is more modest than I expected. Without the correction, new agents should estimate density as lower than it really is, causing them to reproduce more frequently than they should. However, as they build up their own morphogen field and thereby improve their density estimates, they should, on average, recognize that density has gotten too high and increase apoptosis frequency as a result. If successful, this compensation may explain why correcting these early density estimates increases accuracy only slightly. Future efforts could test this hypothesis by counting reproduction and apoptosis events. Such counts could be a useful performance measure anyway, because achieving a result with fewer reproduction and apoptosis events is likely preferable to achieving a similar result with more such events.

The effect on accuracy of correcting new agents' target densities is shown in fig. 5.22b. The benefit of this correction is more dramatic than the density estimate correction. This efficacy is expected. Without correcting new agents' target densities, successive errors can accumulate or multiply, especially near the edge of a tissue, as described in section 3.7.5.



(a) Unsteady-state MSF-SPH correction



(b) Target density correction

Figure 5.22: Accuracy of growth test program, using integral control, with and without corrections for two distinct challenges for a recruitment model of reproduction. (a) shows the benefit of newly recruited agents correcting their MSF-SPH estimates based on the time needed to build to steady state their own contribution to the morphogen field. (b) shows the benefit of newly recruited agents adjusting their parents' target densities to account for their slightly different location in the swarm. $N \geq 20$.

5.11 Demonstrating Mixed Differentiation States

Section 3.6 introduces an approach to allow programmers to describe mixtures of tissue of different differentiation states. Differentiation states are treated as extra dimensions so that continuity, though broken in space, can be maintained in a larger configuration space. I demonstrate this technique with an extension of the plant growth program used above. In addition to the stem growing from the bottom of the domain, a second stem grows from the side, and the two cross. Each starting square is 2 mm on a side, and initial relative density is still 0.1, for a total of around 637 agents at the start. The code describing each stem is similar, with two differences. First, the angle of growth with respect to gravity is different. Second, the two stems have distinct differentiation states, allowing each subtissue to ignore the other even while the meristems cross paths. For simplicity, in this program I assume agents are already differentiated. (The actual process of differentiating in response to an environmental cue is, in my simple model, trivial; adding it to this program would mean adding two lines of the form $\nabla A = 0.1[\text{CUE}(A) > 0](1 - A)$.)

substance twoPlantStems :

scalar fields :

ρ_A // Mass density of stem in state A

ρ_B // Mass density of stem in state B

C_A // Color field for stem in state A

C_B // Color field for stem in state B

T // Timer to delay growth until meristems form

M_A // Meristem for stem in state A

M_B // Meristem for stem in state B

vector fields :

$\mathbf{V}$ // Tissue velocity

behavior :

$C_A = 1$

$C_B = 1$

```

DT = 1

Γ = [T > 10800]1

DρA = 0

DρB = 0

let βA = [EDGEDETECTCOLOR(ρA) > 0.3 - 0.1Γ]

let βB = [EDGEDETECTCOLOR(ρB) > 0.3 - 0.1Γ]

let γA = cos-1  $\left( \frac{\mathbf{g} \cdot \nabla C_A}{\|\mathbf{g}\| \|\nabla C_A\|} \right)$ 

let γB = cos-1  $\left( \frac{R_{\pi/2} \mathbf{g} \cdot \nabla C_B}{\|\mathbf{g}\| \|\nabla C_B\|} \right)$ 

DMA = [A > 0.5](0.1[γA < 0.4π ∧ βA = 1](1 - MA)
      - 0.1[γA > 0.6π ∧ βA = 1]MA
      + 0.02[βA = 0 ∨ 0.4π ≤ γA ≤ 0.6π]∇2MA)

DMB = [B > 0.5](0.1[γB < 0.4π ∧ βB = 1](1 - MB)
      - 0.1[γB > 0.6π ∧ βB = 1]MB
      + 0.02[βB = 0 ∨ 0.4π ≤ γB ≤ 0.6π]∇2MB)

let V' = -0.0001[A > 0.5]ΓMAg - 0.0004[βA = 0]∇ρA
      - 0.0001[B > 0.5]ΓMBRπ/2g - 0.0004[βB = 0]∇ρB

let Vmax = 0.01

V = Vmax tanh  $\left( \frac{\|V'\|}{V_{\max}} \right) \frac{V'}{\|V'\|}$ 

```

Results are shown in fig. 5.23, and a video (growxdemo.mp4) is attached. As expected, there is little interaction between the two stems of distinct differentiation state. Overall density ρ is twice as high in the central square where the two stems overlap, but densities with respect to differentiation state A and B (that is, ρ_A and ρ_B) are unaffected by the overlap. This simple program does not make use of the overall density ρ or its spatial derivatives, but that information is also available to the tissue for use in more complex Morphogen programs.

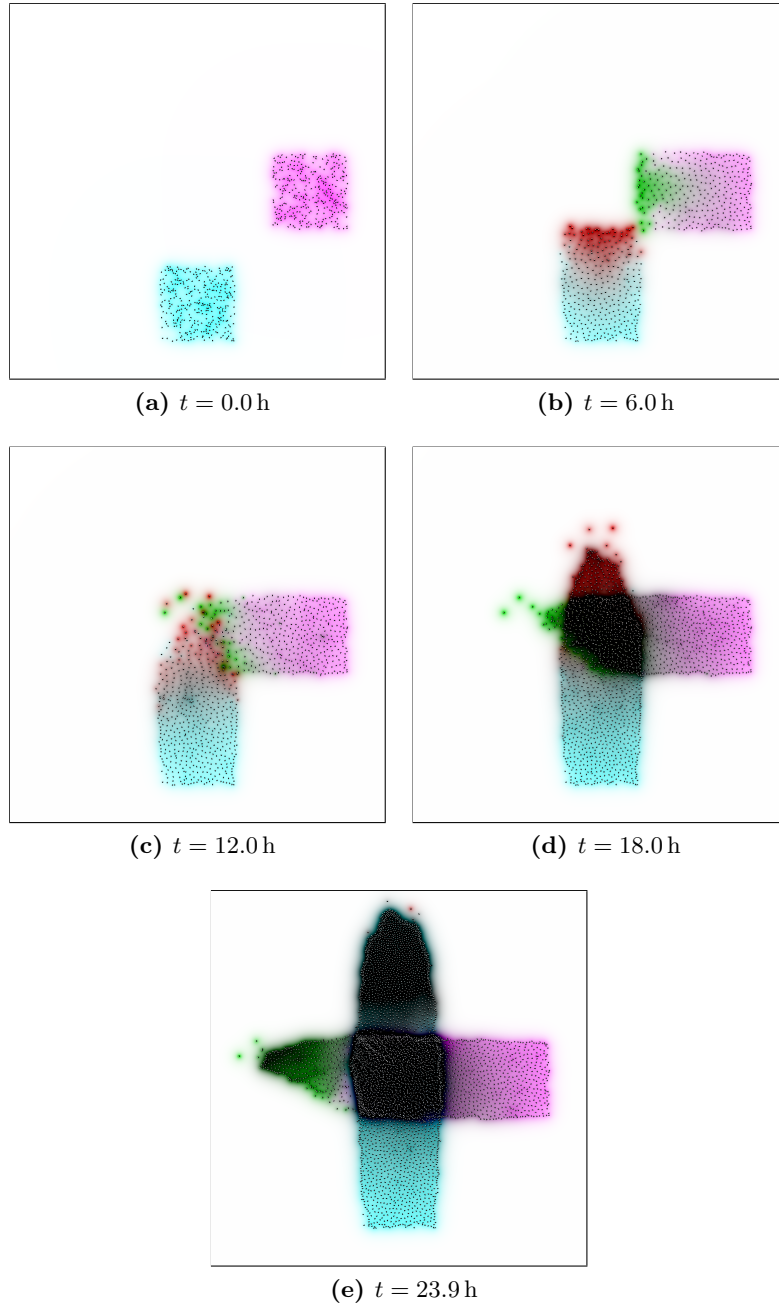


Figure 5.23: Demonstration of mixtures with two crossing stem-like structures. Red and green represent the meristem morphogens with respect to each of the two differentiation states. Cyan and magenta likewise represent the two density morphogens with respect to each differentiation states.

5.12 Edge Finding

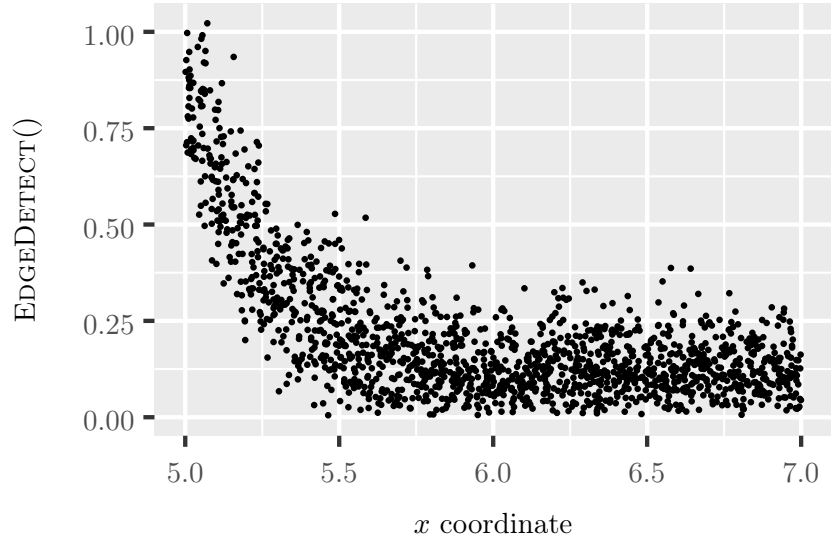
Section 3.8 describes two methods for identifying the edge of tissue. Here I test each with a static field of agents. Agents are randomly placed in the right half of a square domain 10mm on a side up to a relative density of 0.1 (around 3979 agents). For these tests, there is no Morphgen program except to set the value of a color field c to one everywhere. I wait for the resulting two morphogen fields (one for mass density, one for the color field) to reach approximate equilibrium, and then I record the values at each agent for `EDGEDETECT()` and `EDGEDETECTCOLOR(c)`.

Figure 5.24 plots the value of each function compared to each agent's x coordinate, for x coordinates between 5 and 7 (there is no appreciable change beyond $x = 7$). As expected, both functions reach a value of approximately 1.0 for agents very close to the edge at $x = 5$. In addition, both functions remain low farther from the edge, which is a welcome result but not what I had expected. From my experience using each type of edge finding as part of larger Morphgen programs, it seems that the mass density approach is more likely to lead to false positives than the color field approach. For this reason, I had expected to see occasional spikes in these plots even well away from $x = 5$, which are not apparent. Although the uniform random placement I have used does result in some sparse or dense local areas, these fluctuations may not be an adequate model of density fluctuations in more complex programs.

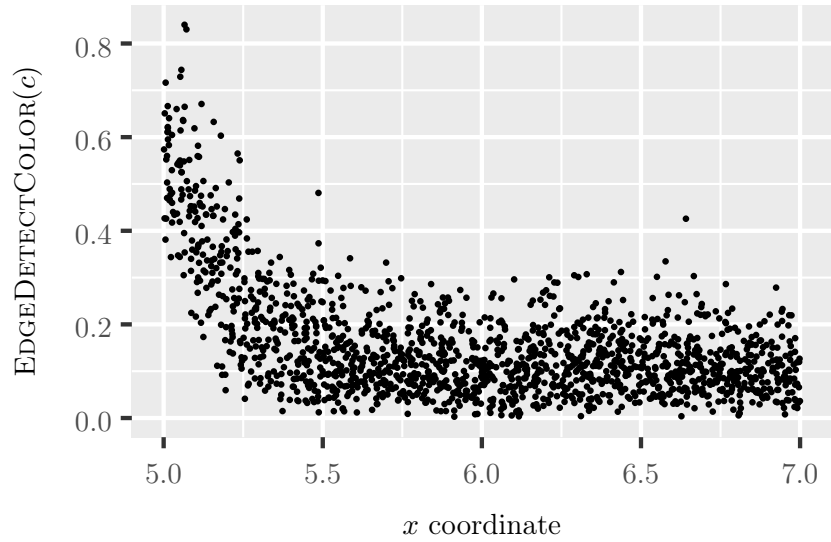
A second difference between the density and color field approaches to edge finding is more clear from fig. 5.24: the edge as defined by the color field approach is sharper than the edge as defined by the density gradient approach. This makes sense because the color field, loosely speaking, attempts to even out variations in the density field as much as possible. So it should maintain a smaller gradient closer to the edge than the density field itself. Whether this difference favors use of the density gradient approach or the color field approach may depend on the nature of a given Morphgen program and the motivation for identifying an edge.

5.13 Path Finding

In this section I demonstrate a more complex Morphgen program in which tissue converges toward the shortest path between two square patches containing distinct environmental cues. Such forming of connections is an important process in morphogenesis. These cues do not diffuse: only tissue located directly on one of the squares can sense its cue. Conceptually, the program proceeds in four steps, although the equations representing these steps are simultaneous:



(a)



(b)

Figure 5.24: Comparison of approaches for edge finding. Figure 5.24a shows each agent's value for `DETECTEDGE()`, which returns the ratio of $\|\langle \nabla \rho(\mathbf{x}) \rangle\| / \langle \rho(\mathbf{x}) \rangle$ to its expected value at an edge, $\sqrt{k/E}$. Figure 5.24b shows each agent's value for `DETECTEDGECOLOR(c)`, which returns the ratio of $\|\langle \nabla c \rangle\|$ to its expected value at an edge, $\sqrt{k/E}$.

- Tissue establishes a gradation in a “seeker” (S) field, outward from cue A .
- Tissue establishes a dynamic path from cue B back to cue A , in a field called “tracer” (T).
- Tissue establishes a relatively stable gradation toward this dynamic path with a “marker” (M) field.
- Tissue concentrates its mass around the established path.

The Morphgen program follows:

cues :

A // Endpoint for path

B // Other endpoint for path

substance pathFinder :

scalar fields :

ρ // Mass density

S // “Seeker,” forms gradation outward from A

T // “Tracer,” forms dynamic path from B back to A

L // “Lock,” recent maximum value of T

M // “Marker,” stabilize and communicate path

vector fields :

$\mathbf{V}$ // Tissue velocity

behavior :

let $\alpha = [L \geq 1 \times 10^{-7}]1$

let $\beta(s) = \frac{2}{1-s} - 1$

let $\text{EXPBLEND}(x, t, s) = [x > 1] + [0 < x \leq t] \frac{x^{\beta(s)}}{t^{\beta(s)-1}} + [t < x < 1] \left(1 - \frac{(1-x)^{\beta(s)}}{(1-t)^{\beta(s)-1}} \right)$

let $\gamma = \text{EXPBLEND}(-10000 \nabla S \cdot \nabla M, 0.5, 0.9)$

let $\epsilon = 1 - \text{EXPBLEND}(\text{EDGEDETECT}(), 0.5, 0.8)$

let $\zeta = \text{VELDOTGRADUPSTREAM} \left(T, 0.003 \frac{\nabla S}{\|\nabla S\|} + \mathbf{V} \right)$

$$\begin{aligned}
\mathbb{D}S &= 0.1[A > 0](1 - S) \\
&\quad + [A = 0] (0.027\nabla^2 S - 0.0002S) \\
\mathbb{D}T &= 0.1[B > 0 \wedge S > 0](1 - T) \\
&\quad + [B = 0 \wedge \|\nabla S\| > 0 \wedge S > 0]\zeta \\
&\quad + 0.1[T > 1](1 - T) \\
\mathbb{D}L &= 0.1[T \geq L](T - L) \\
&\quad - 0.000002[T < L]L \\
\mathbb{D}M &= 0.1[L \geq M](L - M) \\
&\quad + [L < M] (0.00027\nabla^2 M - 0.00009M) \\
&\quad + 0.1[M > 1](1 - M) \\
\mathbf{V} &= -0.0001(1 - \alpha)\|\nabla\rho\| + \alpha\frac{\nabla M}{M} \\
\text{VELOCITYGUARD} &= [A = 0]\alpha (0.92 - (0.92 - 0.99)\text{EXPBLEND}(M, 0, 0.9)) \\
&\quad + 0.999[A > 0] \\
\text{DENSITYGUARD} &= [A = 0]\alpha \exp((0.05 - 0.01) \max(\gamma, \epsilon) - 0.05) \\
&\quad + 0.997[A > 0]
\end{aligned}$$

The lines beginning with keyword **let** are simply convenient macro definitions. The α macro is one if at least some information about the path has spread throughout the tissue. It is zero before such information is present, and will eventually become zero again if path information is lost. Tissue in these states will spread out in hope of finding the missing cue or reestablishing communication with other agents sensing that cue. The `EXPBLEND` macro (along with its helper macro β) is a generalization of the `SmoothStep` clamped sigmoid function with additional threshold and steepness parameters [29]. The purpose of macro γ is, loosely, to determine the extent to which the gradients of S and M are strong and in opposite directions, as a proxy for the location of the path. Macro ϵ indicates not being at an edge.

The $\mathbb{D}T$ instruction, along with the ζ macro, extend the region where T is high along the shortest path. Taken on its own, it is an instruction to advect whatever pattern is found in the T field up the S gradient (toward cue A). But because T is held high at cue B by the agents located there, the effect is to elongate the region where T is high, extending it toward A . At cue B , the gradient of S should point only toward the shortest path, which is why this instruction prevents, at

least in principle, the T field from forming any paths between A and B other than the shortest path. In practice, multiple paths may form if the next-shortest path is too close in length to the shortest path. (This is why I added the crossbar to the T-shaped obstacle: to lengthen the next-shortest path).

T is a rather volatile field, fluctuating with time in the region of the path. Furthermore, its gradient away from the path may not be meaningful. For both these reasons, T is not by itself a robust tool in establishing a velocity field to guide tissue mass toward the shortest path. The L and M fields help transform the information in the T field into forms more useful in informing velocity. L is what I think of as an “internal field”—a field whose spatial derivatives are never referenced in the right-hand side of a Morphogen equation, and which therefore does not need to be communicated among agents or associated with a morphogen. Its purpose is to remember the highest recent local values of the T field, to even out fluctuations. The purpose of M is to diffuse this information away from the shortest path found by the T field, by establishing a gradation throughout the tissue that can be followed to reach the shortest path. The assignment of $\mathbf{V}$ does this job when α is high, following the M gradient toward the regions where peak T values over time are highest.

The $\mathcal{DT}$ instruction seems to be the most delicate part of this program, so accuracy must be high in the region where the path should be formed. The `VELOCITYGUARD` instruction prevents excessive relative motion among agents along the path, using M to determine the region of the path. (Conversely, if M becomes mistakenly high somewhere other than along the intended path, higher accuracy helps tissue correct this error, ultimately decreasing M .) The `DENSITYGUARD` instruction shares this purpose, though it uses γ as a proxy for the region of the path instead of M . The γ macro seems to do a better job of locating potential path regions in addition to the places where M is already high, so basing `DENSITYGUARD` on γ helps prevent agents from prematurely leaving an area before the path has a chance to form. `DENSITYGUARD` is also based on ϵ , allowing higher density gradients around the edge of tissue. This relationship helps tissue converge more fully to the region of the path.

This morphogen program was tested, unchanged, across a 16-fold difference in agent diameter. Initial relative density was held constant at 0.03 across agent diameters, so that agent number density varies 256-fold. With the exception of free morphogen degradation rate, which I consider below, other model parameters are held constant and are shown in table 5.3. Results, shown in fig. 5.25 and in an attached video (pathdemo.mp4), are mixed. The program is successful to varying degrees for the larger four diameters, with the path-finding a bit more aggressive than desired for

Table 5.3: Parameters for path finding program

Parameter	Value(s)					
Agent diameter	20 μm	40 μm	80 μm	160 μm	320 μm	
Timestep	1 s	1 s	1 s	1 s	1 s	
Morphogen half-life (density)	0.5 min	2 min	8 min	32 min	128 min	
Morphogen half-life (other fields)	16 min	16 min	16 min	32 min	128 min	
Morphogen diffusivity			100 $\mu\text{m}^2 \text{s}^{-1}$			
Pixels per millimeter in each direction			75			
Agent body eccentricity			0			
Initial relative density			0.03			
Sensor association rate			600 $\text{M}^{-1} \text{s}^{-1}$			
Sensor association half-life			1 min			

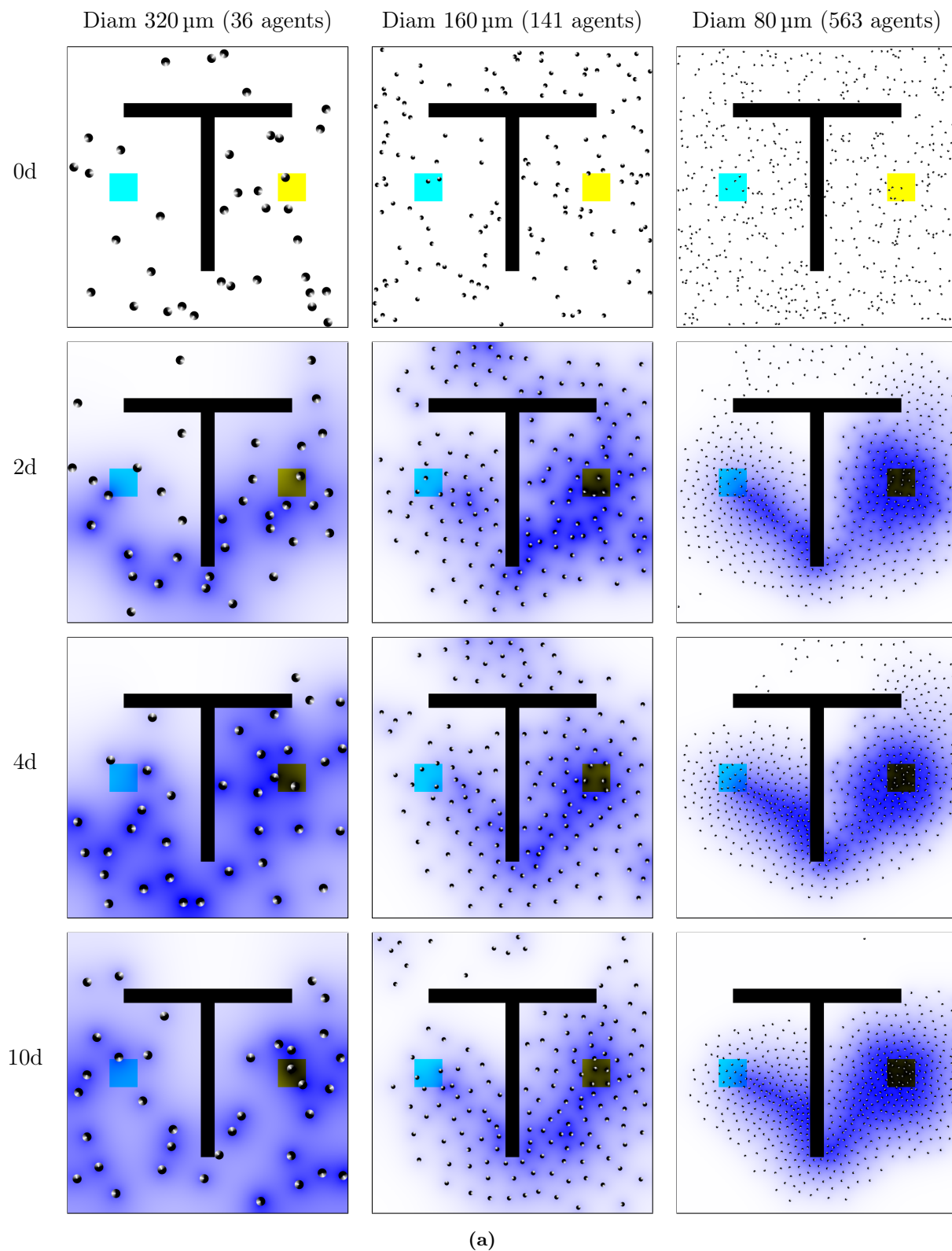


Figure 5.25: Simulations of path-finding Morphogen program for a range of agent sizes. Times in days are given for each row. Cues A and B are shown in cyan and yellow. The morphogen for field M is shown in blue, scaled so that the concentration representing value 1.0 is at full saturation.

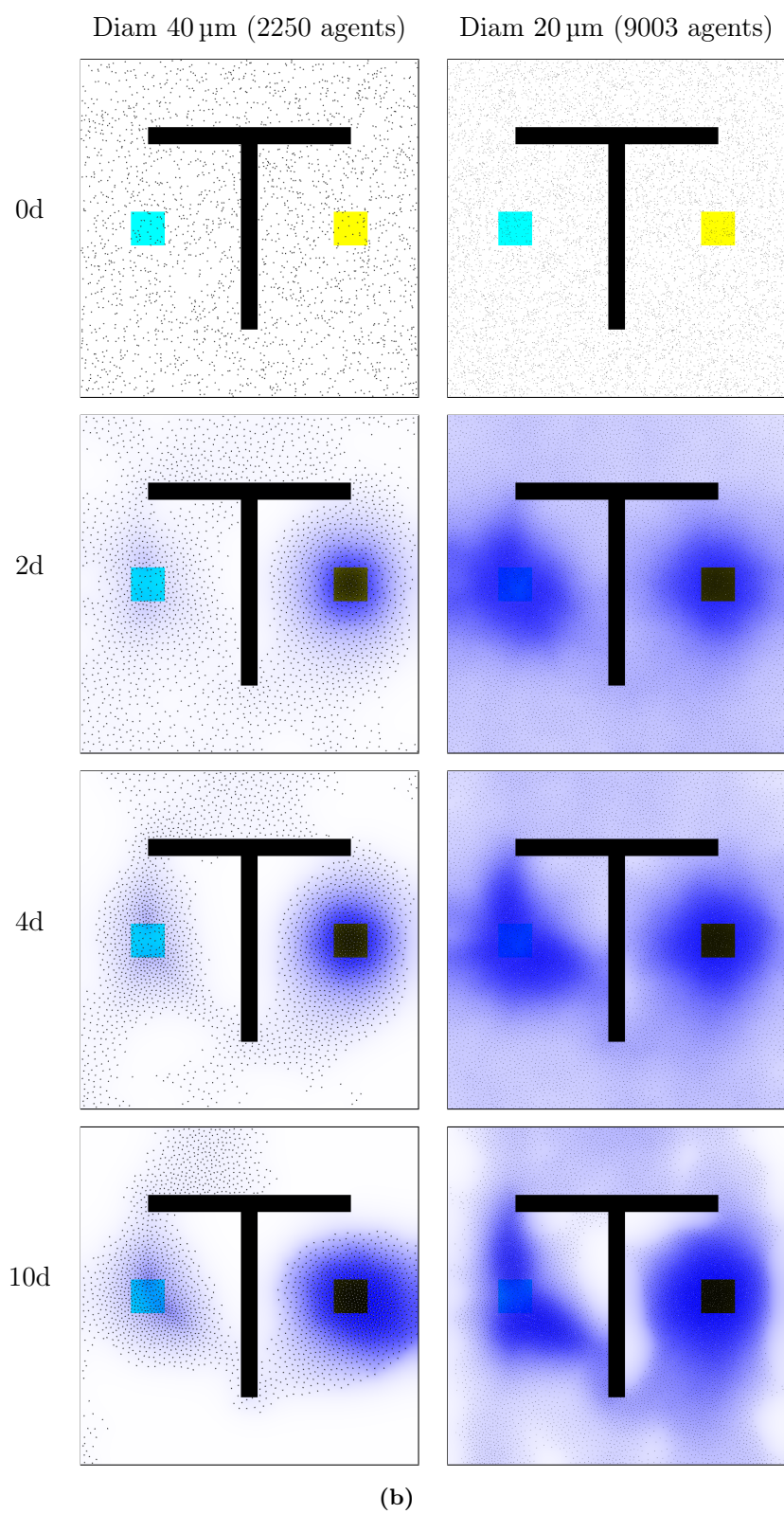


Figure 5.25: (continued)

diameter $160\mu\text{m}$ and a bit weaker than desired for diameter $40\mu\text{m}$. For the smallest diameter, path-finding is too aggressive, which, when combined with `VELOCITYGUARD`, slows agents down such that they have not converged to the path even after ten days.

This imperfect success partially demonstrates the scalability that is a key motivation for my work: MSF-SPH compilation can target any size of agent in its translation of Morphgen’s continuum-based descriptions. I attribute the varying degrees of success at different scales to the challenge of finding an advection rate for the T field that works for all agent densities. (As shown in the code, this rate is 0.003 for the simulations shown.) Because of the Lagrangian nature of SPH, advection is trivial when it can be satisfied by the motion of agents, but difficult when a pattern is to be advected independently of agent motion (section 3.9). If this advection rate is too low, the path formed in field T fails to extend (or extends only weakly, as with diameter $40\mu\text{m}$) all the way to the A cue. But if this rate is too high, T may also form a path along the next-shortest route from A to B (along the top of the T-shaped obstacle), as seen to some extent with diameter $20\mu\text{m}$. From informal testing, there seem to be successful ranges for this parameter with respect to each of the agent diameters considered. Unfortunately, I have yet to demonstrate that there is any one setting for this parameter that works for all of them.

This challenge may have limited importance in practice. It is difficult to imagine a scenario in which one Morphgen program must be compiled across a wide range of agent sizes with no opportunity to fine-tune parameters. Furthermore, the Morphgen program may be considered to be fundamentally the same despite the tuning of one or a few parameters. Nonetheless, my goal with this path-finding program has been to explore the scalability of an unmodified Morphgen program, so I have sought a single parameter set that would work across the widest range of agent sizes. In the future, if I cannot find one Morphgen program that successfully scales with no modifications, I may present results allowing for one or a handful of parameters to be tuned.

Morphogen physical parameters (diffusivity and degradation rate) are, like agent diameter, variable parameters of a target system for compilation. I had therefore hoped to hold these parameters constant across the range of tested agent sizes, to demonstrate that compilation can account for a wide range of ratios of smoothing length to agent size. I was only partly successful in meeting this goal. I was mostly able to hold the morphogen parameters for the S , T , and M fields constant, with an exception I will detail shortly. However, it proved important in practice to scale smoothing length along with agent size for the morphogen corresponding to the density field. For this morphogen, I vary degradation rate to hold constant the ratio of smoothing length to inter-agent

distance. (Because smoothing length is $\sqrt{E/k}$ and number density scales with the inverse square of diameter, this ratio is held constant by dividing degradation rate by four, or quadrupling half-life, with each doubling of agent diameter.) This scaling seems important when a program relies on a relatively even spacing of agents to avoid spurious local reversals in the gradient of another field (as described in section 3.10.1). If the smoothing length of the morphogen associated with the density field gets too large relative to inter-agent distance, agents are unable to resolve variations of high spatial frequency in the density field. Conversely, a too-short smoothing length prevents agents from responding to longer-range variations in the density field; in this case, agents may be evenly spaced but unable to satisfy global density patterns implied by Morphgen code. (The exception to my holding fixed the parameters for the S , T , and M fields is that, for the largest two agent diameters, I increase associated morphogen half-lives from 16 min to match the half-life for the density field. Program performance seems to suffer when the smoothing length for the density field is larger than the smoothing lengths for the other fields, but more investigation is needed to understand why this might be.)

In addition to scalability, this Morphgen program demonstrates that multiple abstract fields can depend on each other to demonstrate complex behavior. Care must be taken, however, when simulating a program in which the behavior of one or more fields is qualitatively similar to physical diffusion and degradation, as in the case of the S field here. In simulating an earlier version of this program, I made the mistake of giving agents excessive sensor precision, allowing all agents in the domain to sense the gradient of the seeker morphogen as secreted by those agents directly in contact with the left square. The program was thus able to work even without the $\nabla^2 S$ term in the Morphgen program.

This observation speaks to the power of directly using physical morphogen gradients in artificial morphogenesis. However, a central focus of my own work is to explore the potential for generating abstract fields atop physical morphogen fields. With this aim in mind, I have kept sensor precision low enough (through the association rate constant) that, at least for simulations with higher agent number density, the Morphgen program fails without the $\nabla^2 S$ term. (Specifically, the S field is never detected at cue B , so the T and M fields never form.) Such failures demonstrate that this Morphgen program genuinely relies on multiple abstract fields working together.

For simulations with larger and fewer agents (specifically, those with agent diameter 160 μm and 320 μm), this condition that the program should rely on the $\nabla^2 S$ term is more challenging to satisfy. Greater distances between agents require larger smoothing lengths, such that the gradient of one

agent's morphogen field is more likely to be detectable to agents on the far side of the domain. Also, I cannot satisfy this condition with respect to the marker field; that is, the program still works even without the $\nabla^2 M$ term. I believe this is because the tracer field (on which the marker field is based) is, in practice, significantly nonzero throughout the domain, so that the marker field is primarily needed to smooth temporal oscillations in the tracer field rather than to communicate it beyond the region of the intended path.

5.14 Crawling vs. Swimming

Having introduced a few demonstration Morphgen programs, I make some key comparisons between the crawling and swimming models. For most of this chapter I have used agents of diameter $40\text{ }\mu\text{m}$, which are sufficiently small that most simulations involve several hundred to a few thousand agents. To make swimming simulation times manageable, in this section I use agents of diameter $80\text{ }\mu\text{m}$ or $160\text{ }\mu\text{m}$. I use the same initial mass densities as elsewhere in this chapter, so that fewer agents are used and results therefore tend to be somewhat less true to the continuum solutions for the Morphgen programs.

Figure 5.26 compares accuracy for the heat equation test program from section 5.1 with the swimming vs. crawling models, both with and without the additional lateral motion introduced in section 5.4.1. Parameters are the same as in table 5.1, except for agent diameter, timestep, and grid resolution, which are $160\text{ }\mu\text{m}$, 2.13 s , and 37.5 px mm^{-1} , respectively. For simulations with lateral motion, although boundary conditions are periodic in the x -direction with respect to agent position and diffusion, they are not periodic with respect to fluid velocity; instead, no-slip boundary conditions are applied. Comparisons for fig. 5.26a are made, as elsewhere in this chapter, with ideal results obtained by FDM simulation. For fig. 5.26b, comparisons are also made with these same FDM results. (The FDM simulation does not include the lateral motion addition; this works because the lateral motion returns the center of mass to its starting position by the end of the simulation.) These latter comparisons are different from before, when comparisons were made with averaged MSF-SPH simulations, because in the present case we are not testing noise-type parameters and so should not assume that differences between the crawling and swimming models represent inaccuracy.

Figure 5.27 compares the crawling and swimming models with the advection program used in section 5.7 to test the upwind scheme. Parameters are as in table 5.1, except for agent diameter and timestep, which are $80\text{ }\mu\text{m}$ and 0.533 s respectively. Boundary conditions are as described

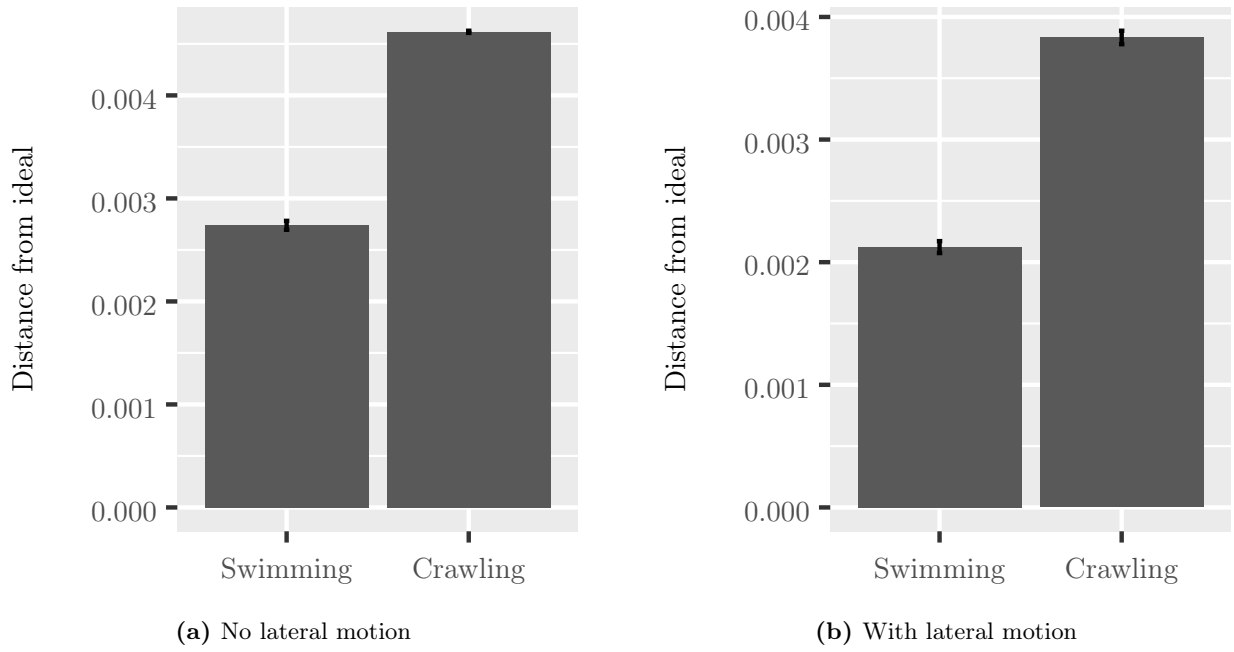


Figure 5.26: Accuracy of swimming and crawling models with two heat equation test programs. $N \geq 12$.

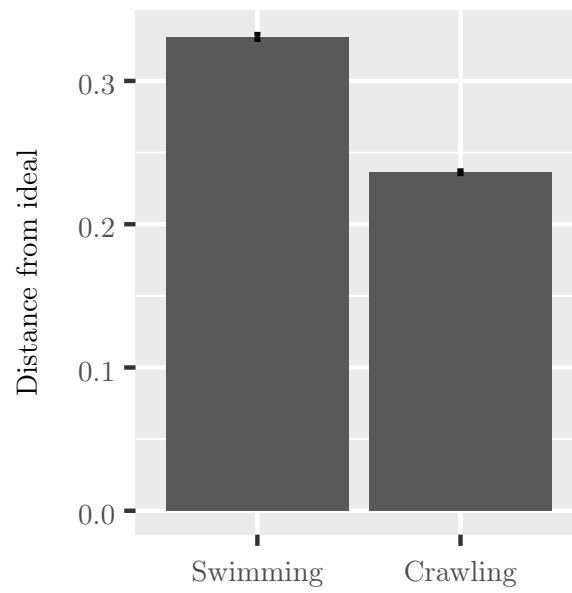


Figure 5.27: Accuracy of swimming and crawling models with advection test programs. $N = 39$.

above for the heat equation test program with lateral motion. Figure 5.28 compares the crawling and swimming models with the plant growth test program from section 5.8. Parameters are as in table 5.2, except for agent diameter, timestep, and grid resolution, which are $160\text{ }\mu\text{m}$, 2.133 s , and 37.5 px mm^{-1} , respectively. For these latter two programs, as is the case elsewhere in this chapter where they are used, comparisons are made with simple geometric calculations of what ideal results should look like: an unchanged Gaussian pattern in the first case, a rectangle of appropriate height in the second.

Finally, fig. 5.29 shows a visual comparison of a path-finding simulation with crawling vs. swimming. The two simulations use the same random seed, so that initial agent placement is the same between them. Parameters are the same as given in table 5.3 for agents of diameter $160\text{ }\mu\text{m}$, except for timestep and grid resolution, which are 2.13 s and 37.5 px mm^{-1} , respectively. The levels of the M field are higher with the swimming model, but because it is the gradient of M that is most important, the more significant difference is that with the crawling model M is higher near cue B , whereas with the swimming model M is more balanced between the two. However, in both cases the Morphgen program is qualitatively successful, with agents converging to the shortest path.

These results suggest that the flow of medium and morphogens around agent bodies as they move does not qualitatively affect the ability of MSF-SPH to successfully implement Morphgen programs. Future work can more thoroughly explore whether the various parameters discussed in this chapter have similar relationships with accuracy for the swimming model as they do for the crawling model. Such tests will be time-consuming but straightforward.

I suspect one common explanation for the apparent increased accuracy of the swimming model with respect to the two heat equation test programs, for the decreased accuracy of the swimming model with respect to the advection and plant growth programs, and for the stronger M field for the swimming model in the path-finding program. Specifically, even without agent motion, the opacity of agent bodies to diffusing morphogens in the swimming model should exaggerate gradient estimates by increasing the path length from one side of an agent to the other. For example, if all morphogen were diffusing from a point on one side of an agent, the difference in concentration from that side of the agent to the other should be larger than the concentration difference across the same span in some region away from the agent, where diffusion is unimpeded. This hypothesis could be tested in the future by keeping the fluid velocity model in place but making agents transparent to diffusion; if the hypothesis is correct, this (unrealistic) swimming model variant should mostly

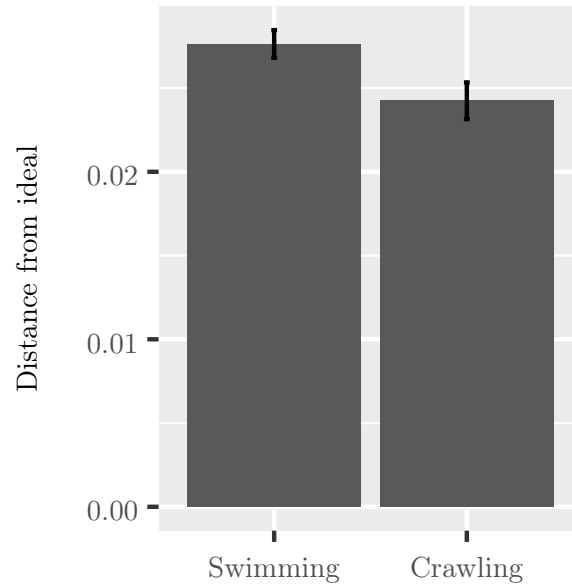


Figure 5.28: Accuracy of swimming and crawling models with plant growth test programs. $N \geq 18$.

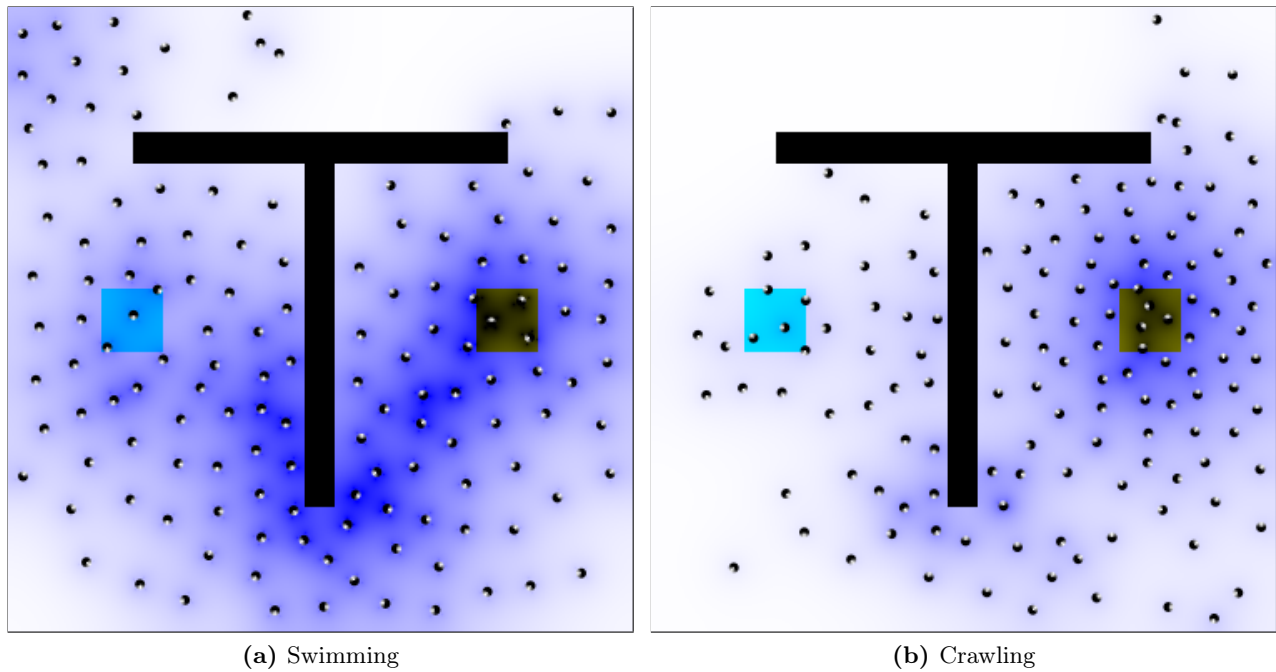


Figure 5.29: Swimming vs. crawling in a path-finding simulation. Each simulation is shown after ten days. The morphogen for field M is shown in blue, scaled so that the concentration representing value 1.0 is at full saturation.

eliminate differences in accuracy as compared to the crawling model. In addition, I would expect the exaggeration of gradient estimates to increase with agent density, as free space for diffusion around a given agent decreases. (At very high densities, a swarm of agents might come to resemble a soil or mineral matrix. Effective diffusivity in such matrices is governed by “porosity,” a measure of the amount of free fluid space relative to solids.)

That exaggerated morphogen gradient estimates would hurt accuracy in the latter two comparisons (advection and plant growth) is unsurprising. The apparent increase in accuracy in the first two (heat equation) comparisons may be explained by the same effect described in [section 5.4.1](#): exaggerated gradient estimates would cause agents to move outward faster, potentially balancing overly slow motion due to lags in morphogen dynamics. Finally, exaggerated morphogen gradient estimates would be expected to increase the value of the ∇T advection term in the path-finding program, explaining the stronger M field there.

If my hypothesis is correct, future work may reveal ways to correct for this problem. For example, it may be possible to employ a second type of calibration, in which estimated and expected gradients are compared as a function of agent size, agent density, and morphogen free diffusivity. Alternatively, it might be possible to analytically derive the expected degree of exaggerated gradient estimates, based on these three parameters. If some such correction can be made, the corrected swimming model may even prove superior to the crawling model, because an exaggeration of concentration differences may allow agents to be more sensitive to subtle gradients.

Chapter 6

Concluding Discussion

Chapter 3 details my research contributions, the most important of which are

- a description of a global-to-local compiler for Morphogen code, summarized in section 3.11, and
- the use of natural smoothing functions (NSFs) for an embodied and naturally parallel communication method for SPH robotics, with radiation- and morphogen-based instances (RSFs and MSFs), described in sections 3.1 to 3.3.

In addition to these two primary contributions, chapter 3 describes several other secondary contributions:

- A correction term for non-steady-state inaccuracies in MSF-SPH Laplacian estimates (section 3.4.1).
- A calibration approach to correcting for non-steady-state inaccuracies in MSF-SPH field value and gradient estimates (section 3.4.2).
- Allowance for spatial mixtures of subtissues with distinct properties by treating differentiation states as extra dimensions in a configuration space (section 3.6).
- Two methods to allow density fields to be specified independently of velocity divergence, assuming agents with abilities analogous to reproduction and apoptosis (section 3.7).
- Two methods (one involving density fields and the other involving color fields) for defining and detecting tissue edges without the need for hard-wired tissue densities (section 3.8).

- An upwind scheme compatible with NSF-SPH (section 3.9).
- Scalable methods for maintaining desired levels of “smoothness” in density and velocity fields (section 3.10).

Chapters 4 and 5 focus on the practicality of MSF-SPH, the morphogen-based version of NSF-SPH. A simulation model is used to examine expected limitations on the accuracy of MSF-SPH systems, and to evaluate some of the methods presented in chapter 3 for overcoming these obstacles. I conclude that MSF-SPH is a plausible future technology, though challenges remains before it can be implemented. But my work has more immediate implications: RSF-SPH is within the realm of current technology, and a traditional SPH implementation of Morphgen compilation can be implemented immediately in robotic swarms.

The present chapter discusses some near-future implications of my research. I outline possibilities for school classrooms. I briefly address specific ethical implications, and I identify broader, related ethical concerns. Finally, I identify key general areas for future research.

6.1 Education

My research opens new possibilities for computer science education. Students will be able to write their own Morphgen code and implement it with NSF-SPH, both in simulation and using physical robots. This experience will provide students with opportunities to deepen their understanding of differential equations and to introduce them to ideas in analog computing, self-organization, and embodiment.

6.1.1 Speculative Classroom Implementation

Students will be able to use the simulator immediately in the manner that I have been using it. However, additional work to add a graphical settings tool and a Stage I compiler (discussed further in section 6.3) would make the simulator more accessible to students. A more ambitious goal would be a physical implementation of NSF-SPH. Although an MSF-SPH implementation is not currently feasible, I believe a cost-effective, demonstration-scale RSF-SPH system could be constructed using current technology.

The primary challenge in implementing RSF-SPH at classroom scale is that smoothing functions would tend to be too large when relying on atmospheric attenuation of light. Visibility in even the

thickest natural fog is rarely under 10–20 m, which corresponds to the radius within which 95% of the corresponding RSF’s mass would be found—too wide for robots on a desktop. It would be easier to achieve higher attenuation in liquid, but making robots that work underwater would add complexity and cost. I propose a solution in which robots can operate in air, but in which light is attenuated as needed in a solid phase.

Consider a large slab of translucent white plastic for robots to operate on. Each robot would have a multicolored LED light on its underside, nearly flush with the slab. Four light intensity sensors, pointing downward, would be placed around the robot’s periphery. The robot could have a skirt blocking any residual line of sight from the light on its underside to its own or any neighbor’s sensors.

This setup would allow for a uniform light attenuation, with the attenuation coefficient controllable during the manufacturing process. Light would enter the slab beneath a robot; it would then travel through the slab, scattered sideways (and in other directions) by the plastic. After exiting the slab, most light not hitting a sensor would be absorbed by the room’s relatively distant walls and ceiling, more so if the latter are draped in dark fabric. The sensors would not, of course, directly measure scalar irradiance inside the slab, but would measure instead the light escaping upward from the slab due to scattering. Lights in the room would need to be off, which would also serve to dramatize the colored RSF fields.

Because the idealized RSFs described in section 3.2 do not account for scattering, the RSFs implied by the above procedure will have somewhat different shapes than presented in that section. But as mentioned there, these RSFs will still be valid smoothing functions. It would be interesting to derive the expected shape of RSFs implied by media with both scattering and absorption. However, the simplest way to find the implied RSFs would be to build the system, have one robot emit light at some arbitrary power, and examine neighbors’ sensor readings at various distances. After numerical integration, the same procedure as in section 3.2 would then be used to relate source power to field value.

This approach would have the advantage that the RSF fields would be visible as colors on the surface of the plastic slab. A disadvantage would be that each slab would have a smoothing length baked in at manufacture time. To demonstrate the effect of different smoothing lengths, educators could keep two or three slabs on hand. Alternatively, the slab could be replaced with a shallow tank that could be filled with liquid solutions of varying opacity. This tank could be fully enclosed,

with robots atop a clear lid, or it could be open, if robots are designed to operate with their wheels, lights, and sensors submerged.

Apart from lights and sensors, robots used in this demonstration would otherwise be similar in spirit to others used in swarm robotic education, such as e-pucks [85] or Kilobots [127]. The other key difference with such robots would be that these RSF-SPH robots would not need to establish data links with neighbors, reducing complexity and cost. (This lack of need for individual-based communication is, of course, a motivating advantage of NSF-SPH.)

6.1.2 Educational Benefits

NSF-SPH Morphgen compilation will deepen students' understanding of what computing and programming can look like. For decades, proponents have argued for better inclusion of analog computing in computer science education [84, 32]. NSF-SPH Morphgen compilation represents a new opportunity to teach analog computing concepts. As with traditional analog computing education, Morphgen compilation will allow students to interact with differential equations intimately, gaining an intuitive understanding of their behavior to complement the more symbolic orientation of typical math instruction. But in this regard I believe Morphgen compilation has advantages over traditional analog computation.

First, Morphgen compilation enriches the computing environment with space and parallelism. In Morphgen space is not an abstraction, as in an analog computer; rather, space is an integral part of the computing domain. And agent-based compilation of Morphgen introduces multiple analog elements interacting in this spatial domain.

Second, whereas traditional analog computing focuses on modeling other systems, Morphgen programs (in the context of my work) are self-constructing systems; the processes are ends in themselves. There are no wrong Morphgen programs, only those interesting or useful. Creativity is, of course, important in any type of computing; however, Morphgen gives students the opportunity to create something more deeply their own than when modeling existing systems.

This advantage of Morphgen compilation in bringing a new kind of creativity to analog computing has a parallel with respect to self-organization. Self-organization is a concept generally taught in reference to systems over which students have little individual control. Compiling Morphgen code gives students an opportunity to design their own self-organizing systems.

NSF-SPH Morphgen compilation in particular will give students experience with embodied computation, with a system that relies on its own physical nature for crucial information processing.

At least in simulation, students will be able to vary parameters of physical NSF fields and observe the effect of these changes on computation.

Neither the ability for students to design their own self-organizing systems nor the opportunity to study embodied computation are by themselves unique to NSF-SPH Morphgen compilation, but are shared by other topics in bio-inspired computing. However, I believe this combination of traits displayed by NSF-SPH implementation of Morphgen—analogue, parallel, spatial, self-organizing, and embodied computation—position it well to broaden students’ understanding of computing.

6.2 Ethics

Ethical questions related to my research can be divided into two categories. First, what dangers specifically attend the new ideas introduced? Second, what dangers more generally attend the broader fields incrementally advanced by my research? Identifying some of these concerns may contribute toward policy that anticipates future technological advances.

Regarding the first category of ethical concern, I do not believe that my research poses significant dangers specific to the new ideas I introduce. The most qualitatively new technology I describe is the use of NSF-SPH to control agent swarms. This technology seems unlikely to cause harm either through its intended function or through the subversion of that function; I address each in turn.

Compared to other swarm control approaches, NSF-SPH systems may be less likely to cause harm through their intended function, because they appear to be inherently insecure. In order for NSF fields to sum naturally in the environment, they must be openly accessible. In anthropomorphic terms, nature itself needs access to the information represented by an NSF field in order to sum it. An attacker could therefore jam an NSF-SPH swarm simply by flooding its working domain with whatever NSF fields the swarm is using.

Because of this vulnerability to attack, an NSF-SPH system could cause harm through the subversion of its intended function. However, because this vulnerability appears to be both obvious and intrinsic, my expectation is that NSF-SPH will be used only where a locally friendly environment can be established. In cases where the environment’s security might be in question, improving it (e.g., through some form of shielding) is beyond the scope of my research.

Addressing in detail the second set of ethical concerns (those attending generally to fields incrementally advanced by my research) is also beyond the scope of my research. However, I identify a few key concerns related to robot swarms and to synthetic biology.

The idea of using SPH as a basis for compiling a language such as Morphogen (section 3.5) applies even to traditional swarm robotics, not just NSF-SPH systems. Therefore, this technique could be used in controlling a wide range of macrorobotic swarms, including those used by militaries. Militaries are developing swarm robotic systems for offensive, defensive, and support roles, but the legal and ethical implications of this technology are not always fully appreciated [156].

One practical concern with military swarms is proliferation. The possibility that swarms may be a good defense against enemy swarms raises the specter of an arms race among militaries [63]. Also of concern is the proliferation of swarm technology to violent nonstate actors (VNSAs), including terrorists. An added concern with VNSAs is that proliferation may occur independently of developments in military technology: regarding drones in general (but including drone swarms specifically), Chávez and Swed [15] find that civilian models are more attractive to VNSAs than military versions because of the former versions' low cost, small size, and ease of use.

Other concerns relating in particular to offensive military swarms involve issues of responsibility. As with any potentially autonomous weapon, swarms raise the questions of who would be responsible for lethal decisions made without meaningful human control [159] as well as whether it is inherently impermissible to automate the decision to kill a human [63]. Current US military policy is for human soldiers to make any final decision on use of lethal force [63]. However, there is concern that the burden of making these decisions, as well as the work of controlling swarm weapons more broadly, may be too cognitively taxing for a human. Humans may therefore shoulder responsibility for lethal decisions that they cannot be expected to make well [156]. In something of a converse, there is also concern that swarms, like drones, may place human decision makers at such remove from the field that they do not feel an appropriate weight of responsibility [156, 159].

Although key aspects of my research apply to traditional robotic swarms, MSF-SPH in particular envisions tiny agents in aqueous environments. Biological cells are not only the inspiration for such agents but also serve as a potential implementation for them, by way of synthetic biology (section 2.5.1). Synthetic biology raises several ethical concerns, both practical and conceptual.

One of the most prominent of practical concerns is that synthetic biology could be deliberately misused to design bioweapons [20]. Bioweapons generally rely on the infectious or toxin-producing functions of individual cells or virions as opposed to arrangements of the same, and are typically not multicellular [101]. Therefore, the respects in which my work may advance synthetic biology are not likely to advance the possibility of using synthetic biology to create bioweapons.

Another practical concern that applies more closely to my work is the possibility that novel life forms could cause harm, especially ecological, by escaping their intended domain [158]. This concern is paralleled by similar possibilities for self-replicating robots [23]. Agents could escape in their designed form or by evolving into some less controllable form. Proposed technical approaches to mitigating these concerns, both for robots and synthetic biological life, often involve engineering agents to require some unusual resource that can be provided in the intended domain but is uncommon outside it [23]. For example, in the biological case, cells can be engineered to use noncanonical nucleobases in their DNA and RNA or noncanonical amino acids [158].

Apart from practical concerns, synthetic biology also raises questions about its inherent morality in two main respects. First is the possibility that creating what could be considered new life forms violates the proper role of humanity in relation to God or to nature [20, 49]. Second is the possibility that by blurring the distinction between machines and life, we risk treating moral agents as machines [23, 20]. One response to the first concern is that synthetic biology is not categorically different from genetic engineering, and that to engineer life is not to create it [20]. A similar response to the second concern is that deciding the moral status of biological life is already difficult, and that the question as it relates to engineered life is not categorically distinct [20].

6.3 Future Work

I have pointed out throughout this dissertation several narrow topics worthy of future research. In this section, I describe in broader terms a few major future research directions.

A mundane but important step is to improve the simulator in three key regards. First, the simulator should output comprehensive information at regular intervals about both fields and agent state, which could later be replayed with a tool along the lines of scientific visualizers such as ParaView or MayaVi. This feature would simplify the debugging of Morphogen programs; for example, the programmer could pick out a single agent visually and examine its state over time. Second, the simulator could be extended to three dimensional space; this would require extensive but straightforward changes. Finally, the simulator could be modified to work with radiative smoothing functions as well as morphogen smoothing functions.

Another logical next step from my research is to more precisely define a relevant subset of Morphogen and to build for it a literal compiler following ideas discussed in section 3.11. This compiler would have the simulator as its target. Because the simulator already implements the

ideas of Stage II and Stage III of my outlined compilation method in an interpretive mode, for now a compiler need only implement Stage I, which is currently performed by hand. As explained in that section, there is little substantive work to be done in this stage, so most of the work involved in writing this first compiler will be straightforward code translation from Morphgen to corresponding C++. A prototype of such a compiler could be created reasonably quickly with the synmac syntax macroprocessor, for which a continuous (that is, not agent-based) compiler of Morphgen to MATLAB already exists [78].

(It would also be possible to strip the simulator of much of its MSF-SPH interpretive logic so that agents are not MSF-SPH-“aware”, but instead equipped only with some basic abilities to set velocity and secretion rates, sense morphogen concentrations, and evolve a set of ODEs. A more elaborate compiler could then be written that would take on all three stages of compilation, targeting this modified simulator. This exercise might be a worthwhile demonstration of the concepts discussed in section 3.11, but otherwise would offer no practical advantage over my current interpretive approach.)

Another important next step is simply to write more Morphgen programs to further explore the possibilities of MSF-SPH, if only in simulation. For example, my growth demonstration is rudimentary; it would be interesting to add branching to make it more plant-like. Another possibility would be a search-and-rescue demonstration in which agents diffuse through a maze until finding one of several targets, condensing around that target and bringing it to a base, then repeating the process. (Some ad-hoc code would be needed to give agents a stylized ability to move a virtual object when enough agents are present.) A third possibility would be to explore the ability for agents to collectively create oscillating fields. Such an ability could serve as a basis for a more elaborate clock-and-wavefront mechanism such as demonstrated in MacLennan and McBride [80] for continuous tissue.

A third major research direction, more ambitious and less straightforward, would be to integrate the present work, which takes mesenchymal tissue as its inspiration, with work that takes its inspiration from epithelial tissue. Other work in GTLC and morphogenetic engineering has focused on epithelial tissue, such as Nagpal’s work on OSL and its compiler (section 2.3.1). Still other work has focused on developing languages to describe processes relating to both mesenchymal and epithelial tissue, such as that by Somogyi et al. on MML (section 2.2). It would represent a great advance to be able to not only describe both tissue types (and related processes) in a unified way, but to compile such descriptions into code for discrete, active agents. A central

challenge in meeting this goal is that the two tissue types are most naturally described with different dimensionality: mesenchyme in terms of density fields over space, and epithelium in terms of surfaces (two-dimensional manifolds) embedded in space. A further generalization along these lines would be to also incorporate curves (one-dimensional manifolds), such as can describe threads or filaments, into the same unified compilation framework.

6.4 Conclusion

Some of the possibilities explored in this work are immediate, such as traditional SPH implementations of Morphgen compilation. Others may be possible in the near future, such as the educational robots described above. But I have mostly focused on implementing Morphgen through MSF-SPH, which is a more distant technological goal. By exploring some of what could be done with the right sort of technology, I hope that my research helps motivate steps toward that goal in electromechanical, chemical, and biological engineering. I also hope that by building some theoretical framework toward engineering mesenchyme-type processes, my research motivates further theoretical work toward a broader framework that unifies a range of morphogenetic processes, both mesenchymal and epithelial. Because of the promise it holds in medical technology, construction, and micro-scale manufacturing, research in morphogenetic engineering warrants robust support.

Bibliography

- [1] Achour, S., Sarpeshkar, R., and Rinard, M. C. (2016). Configuration synthesis for programmable analog devices with Arco. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 177–193, New York, NY, USA. ACM. [103](#)
- [2] Arbuckle, D. J. and Requicha, A. A. G. (2010). Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: Algorithms and simulations. *Autonomous Robots*, 28(2):197–211. [17](#)
- [3] Bachrach, J., Beal, J., and McLurkin, J. (2010). Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, 19(6):825–847. [13](#)
- [4] Bandala, A. A. and Dadios, E. P. (2016). Dynamic aggregation method for target enclosure using smoothed particle hydrodynamics technique: An implementation in quadrotor unmanned aerial vehicles (QUAV) swarm. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 20(1). [28](#)
- [5] Bandala, A. A., Faelden, G. E., Maningo, J. M., Nakano, R. C. S., Vicerra, R. R. P., and Dadios, E. P. (2016). Implementation of varied particle container for Smoothed Particle Hydrodynamics–based aggregation for unmanned aerial vehicle quadrotor swarm. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1715–1720, Daejeon, Korea. IEEE. [28](#), [83](#)
- [6] Barca, J. C. and Sekercioglu, Y. A. (2013). Swarm robotics reviewed. *Robotica*, 31(03):345–359. [14](#)
- [7] Beal, J., Lu, T., and Weiss, R. (2011). Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLOS ONE*, 6(8). [104](#)
- [8] Berman, S., Halász, Á., Hsieh, M. A., and Kumar, V. (2009). Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics*, 25(4):927–937. [18](#)
- [9] Berman, S., Kumar, V., and Nagpal, R. (2011). Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–385, Shanghai, China. IEEE. [16](#)
- [10] Bernardini, F., Gheorghe, M., and Krasnogor, N. (2007). Quorum sensing P systems. *Theoretical Computer Science*, 371(1-2):20–33. [36](#)

- [11] Bongard, J. C. and Pfeifer, R. (2003). Evolving complete agents using artificial ontogeny. In Hara, F. and Pfeifer, R., editors, *Morpho-Functional Machines: The New Species*, pages 237–258, Tokyo. Springer Japan. [6](#)
- [12] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41. [14](#)
- [13] Cao, Y., Yu, W., Ren, W., and Chen, G. (2013). An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics*, 9(1):427–438. [15](#)
- [14] Chang, K.-H., Chang, T.-J., and Wen-Hann Sheu, T. (2017). Development of an upwinding kernel in SPH-SWEs model for 1D trans-critical open channel flows. *Journal of Hydro-environment Research*, 15:13–26. [87](#)
- [15] Chávez, K. and Swed, O. (2021). The proliferation of drones to violent nonstate actors. *Defence Studies*, 21(1):1–24. [187](#)
- [16] Chorin, A. J. (1968). Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22(104):745–762. [210](#)
- [17] Coore, D. (1999). *Botanical Computing : A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. Thesis, Massachusetts Institute of Technology. [12](#), [19](#)
- [18] Degond, P. and Mustieles, F.-J. (1990). A deterministic approximation of diffusion equations using particles. *SIAM Journal on Scientific and Statistical Computing*, 11(2):293–310. [126](#)
- [19] Disset, J., Cussat-Blanc, S., and Duthen, Y. (2014). Self-organization of symbiotic multicellular structures. In *The Fourteenth International Conference on the Synthesis and Simulation of Living Systems - ALIFE 2014*, pages 541–548, New York, United States. The MIT Press. [34](#)
- [20] Douglas, T. and Savulescu, J. (2010). Synthetic biology and the ethics of knowledge. *Journal of Medical Ethics*, 36(11):687–693. [187](#), [188](#)
- [21] Doursat, R. and Sánchez, C. (2014). Growing fine-grained multicellular robots. *Soft Robotics*, 1(2):110–121. [6](#)

- [22] Doursat, R., Sayama, H., and Michel, O. (2013). A review of morphogenetic engineering. *Natural Computing*, 12(4):517–535. [3](#), [12](#), [105](#)
- [23] Eiben, Á. E., Ellers, J., Meynen, G., and Nyholm, S. (2021). Robot evolution: Ethical concerns. *Frontiers in Robotics and AI*, 8. [188](#)
- [24] Elgeti, J., Winkler, R. G., and Gompper, G. (2015). Physics of microswimmers—single particle motion and collective behavior: A review. *Reports on Progress in Physics*, 78(5):056601. [121](#)
- [25] Eren, U. and Açıkmeşe, B. (2017). Velocity field generation for density control of swarms using heat equation and smoothing kernels. *IFAC-PapersOnLine*, 50(1):9405–9411. [23](#), [24](#)
- [26] Eren, U., Demirer, N., and Açıkmeşe, B. (2018). Density-based feedback control for earth orbiting swarms via velocity fields. *IFAC-PapersOnLine*, 51(12):44–49. [24](#)
- [27] Eren, U. and Hansen, E. (2019). Collision-free decentralised density feedback control for swarms. *IFAC-PapersOnLine*, 52(12):364–369. [24](#)
- [28] Erkok, P., Yasa, I. C., Ceylan, H., Yasa, O., Alapan, Y., and Sitti, M. (2019). Mobile Microrobots for Active Therapeutic Delivery. *Advanced Therapeutics*, 2(1):1800064. [3](#), [31](#)
- [29] ExpBlend (n.d.). <https://www.desmos.com/calculator/3zhzwbfrxd>. Accessed March 15, 2022. [170](#)
- [30] Farsad, N., Yilmaz, H. B., Eckford, A., Chae, C., and Guo, W. (2016). A comprehensive survey of recent advancements in molecular communication. *IEEE Communications Surveys Tutorials*, 18(3):1887–1919. [34](#), [35](#)
- [31] Field, R. D., Anandakumaran, P. N., and Sia, S. K. (2019). Soft medical microrobots: Design components and system integration. *Applied Physics Reviews*, page 21. [3](#)
- [32] Fitch, A. L., Iu, H. H. C., and Lu, D. D. C. (2011). An analog computer for electronic engineering education. *IEEE Transactions on Education*, 54(4):550–557. [185](#)
- [33] Frihauf, P. and Krstic, M. (2011). Leader-enabled deployment onto planar curves: A PDE-based approach. *IEEE Transactions on Automatic Control*, 56(8):1791–1806. [15](#)
- [34] Fujiwara, R., Kano, T., and Ishiguro, A. (2014). Self-swarming robots that exploit hydrodynamical interaction. *Advanced Robotics*, pages 1–7. [24](#)

- [35] Garikipati, K. (2017). Perspectives on the mathematics of biological patterning and morphogenesis. *Journal of the Mechanics and Physics of Solids*, 99:192–210. [1](#)
- [36] Gilbert, S. F. (2013). *Developmental Biology, Tenth Edition*. Sinauer Associates, Inc., Sunderland, MA, USA, 10 edition edition. [1](#), [2](#)
- [37] Gingold, R. A. and Monaghan, J. J. (1977). Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389. [19](#)
- [38] Gómez, E. J., Giral, M., and Martínez Sarmiento, F. H. (2016). Collective multi-agent navigation model based on bacterial quorum sensing. *Tecnura*, 20(47):29–38. [36](#)
- [39] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. <http://eigen.tuxfamily.org>. Accessed March 15, 2022. [212](#)
- [40] Hamann, H. (2010). *Space-Time Continuous Models of Swarm Robotic Systems*, volume 9 of *Cognitive Systems Monographs*. Springer, Berlin. [8](#), [14](#), [15](#), [18](#)
- [41] Hamann, H., Valentini, G., and Dorigo, M. (2016). Population coding: A new design paradigm for embodied distributed systems. In Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., and Stützle, T., editors, *Swarm Intelligence*, volume 9882, pages 173–184, Brussels, Belgium. Springer International. [6](#)
- [42] Hassinen, T., Popov, S., Friberg, A. T., and Setälä, T. (2016). Generation of nearly 3D-unpolarized evanescent optical near fields using total internal reflection. *Optics Letters*, 41(13):2942. [51](#)
- [43] Hermle, R. (2017). Projection method for solving stokes flow. *arXiv:1712.02030v2*. [123](#)
- [44] Hodgkinson, B., Lipinski, D., Peng, L., and Mohseni, K. (2014). High resolution atmospheric sensing using UAVs. In Ani Hsieh, M. and Chirikjian, G., editors, *Distributed Autonomous Robotic Systems*, volume 104, pages 31–45. Springer, Berlin. [27](#)
- [45] Hsieh, M. A., Kumar, V., and Chaimowicz, L. (2008). Decentralized controllers for shape generation with robotic swarms. *Robotica*, 26(05). [24](#)

- [46] Huang, C., Lei, J. M., Liu, M. B., and Peng, X. Y. (2016). An improved KGF-SPH with a novel discrete scheme of Laplacian operator for viscous incompressible fluid flows. *International Journal for Numerical Methods in Fluids*, 81(6):377–396. [42](#)
- [47] Jafferis, N. T., Helbling, E. F., Karpelson, M., and Wood, R. J. (2019). Untethered flight of an insect-sized flapping-wing microscale aerial vehicle. *Nature*, 570(7762):491–495. [30](#)
- [48] Jamali, V., Ahmadzadeh, A., Wicke, W., Noel, A., and Schober, R. (2018). Channel modeling for diffusive molecular communication - a tutorial review. *arXiv:1812.05492*. [36](#)
- [49] Kaebnick, G. E., Gusmano, M. K., and Murray, T. H. (2014). The ethics of synthetic biology: Next steps and prior questions. *Hastings Center Report*, 44(S5):S4–S26. [188](#)
- [50] Kajishima, T., Takiguchi, S., Hamasaki, H., and Miyake, Y. (2001). Turbulence structure of particle-laden flow in a vertical plane channel due to vortex shedding. *JSME International Journal Series B Fluids and Thermal Engineering*, 44(4):526–535. [123](#), [210](#), [211](#), [212](#)
- [51] Kang, S., Iaccarino, G., and Moin, P. (2009). Accurate immersed-boundary reconstructions for viscous flow simulations. *AIAA Journal*, 47(7):1750–1760. [210](#)
- [52] Khaldi, B., Harrou, F., Cherif, F., and Sun, Y. (2017). A distance weighted-based approach for self-organized aggregation in robot swarms. In *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, pages 1–6, Boumerdes, Algeria. [29](#)
- [53] Khaldi, B., Harrou, F., Cherif, F., and Sun, Y. (2018). Self-organization in aggregating robot swarms: A DW-KNN topological approach. *Biosystems*, 165:106–121. [29](#)
- [54] Kicheva, A., Bollenbach, T., Wartlick, O., Jülicher, F., and Gonzalez-Gaitan, M. (2012). Investigating the principles of morphogen gradient formation: From tissues to cells. *Current Opinion in Genetics & Development*, 22(6):527–532. [56](#)
- [55] Kicheva, A., Pantazis, P., Bollenbach, T., Kalaidzidis, Y., Bittig, T., Jülicher, F., and González-Gaitán, M. (2007). Kinetics of morphogen gradient formation. *Science*, 315(5811):521–525. [54](#), [56](#)
- [56] Klavins, E., Ghrist, R., and Lipsky, D. (2006). A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962. [17](#)

- [57] Kondacs, A. (2003). Biologically-inspired self-assembly of two-dimensional shapes using global-to-local compilation. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 633–638, Acapulco, Mexico. Morgan Kaufmann. [13](#)
- [58] Kornberg, T. B. (2012). The imperatives of context and contour for morphogen dispersion. *Biophysical Journal*, 103(11):2252–2256. [54](#)
- [59] Koukouvini, P. K., Anagnostopoulos, J. S., and Papantonis, D. E. (2013). An improved MUSCL treatment for the SPH-ALE method: Comparison with the standard SPH method for the jet impingement case. *International Journal for Numerical Methods in Fluids*, 71(9):1152–1177. [87](#)
- [60] Kowaliw, T., Bredeche, N., Chevallier, S., and Doursat, R. (2014). Artificial neurogenesis: An introduction and selective review. In Kowaliw, T., Bredeche, N., and Doursat, R., editors, *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks*, Studies in Computational Intelligence, pages 1–60. Springer, Berlin, Heidelberg. [3](#)
- [61] Kriegman, S., Blackiston, D., Levin, M., and Bongard, J. (2021). Kinematic self-replication in reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 118(49). [71](#)
- [62] Krishnan, V. and Martínez, S. (2018). Distributed control for spatial self-organization of multi-agent swarms. *SIAM Journal on Control and Optimization*, 56(5):3642–3667. [15](#)
- [63] Lachow, I. (2017). The upside and downside of swarming drones. *Bulletin of the Atomic Scientists*, 73(2):96–101. [187](#)
- [64] Lipinski, D. and Mohseni, K. (2010). Cooperative control of a team of unmanned vehicles using smoothed particle hydrodynamics. In *AIAA Guidance, Navigation, and Control Conference*, Toronto, Canada. American Institute of Aeronautics and Astronautics. [27](#)
- [65] Lipinski, D. and Mohseni, K. (2013). Nearly fuel-optimal trajectories for vehicle swarms in open domains with strong background flows. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3837–3842. IEEE. [27](#)
- [66] Lisicki, M., Velho Rodrigues, M. F., Goldstein, R. E., and Lauga, E. (2019). Swimming eukaryotic microorganisms exhibit a universal speed distribution. *eLife*, 8:e44907. [116](#)

- [67] Liu, G. R. and Liu, M. B. (2003). *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*. World Scientific, New Jersey. [20](#), [21](#), [39](#), [54](#), [128](#)
- [68] Liu, M. B. and Liu, G. R. (2010). Smoothed particle hydrodynamics (SPH): An overview and recent developments. *Archives of Computational Methods in Engineering*, 17(1):25–76. [21](#)
- [69] Liu, W. and Wu, C.-Y. (2020). A hybrid LBM-DEM framework with an improved immersed moving boundary method for modelling complex particle-liquid flows involving adhesive particles. *arXiv:1901.09745*. [209](#)
- [70] Lucy, L. B. (1977). A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024. [19](#)
- [71] MacLennan, B. J. (2009). Preliminary development of a formalism for embodied computation and morphogenesis. Technical Report UT-CS-09-644. [8](#)
- [72] MacLennan, B. J. (2010). Models and mechanisms for artificial morphogenesis. In *Natural Computing*, pages 23–33. Springer. [8](#)
- [73] MacLennan, B. J. (2012). Embodied computation: Applying the physics of computation to artificial morphogenesis. *Parallel Processing Letters*, 22(03). [8](#)
- [74] MacLennan, B. J. (2014). Coordinating massive robot swarms. *International Journal of Robotics Applications and Technologies*, 2(2):1–19. [8](#)
- [75] MacLennan, B. J. (2015). The Morphogenetic Path to Programmable Matter. *Proceedings of the IEEE*, 103(7):1226–1232. [1](#), [2](#)
- [76] MacLennan, B. J. (2018a). Coordinating swarms of microscopic agents to assemble complex structures. In Ying Tan, editor, *Swarm Intelligence - Volume 1: Principles, Current Algorithms and Methods*, pages 583–612. Institution of Engineering and Technology. [8](#)
- [77] MacLennan, B. J. (2018b). Path creation by continuous flocking as an example of a morphogenetic programming language. *Faculty Publications and Other Works – EECS*, page 24. [8](#)
- [78] MacLennan, B. J. (2018c). The synmac syntax macroprocessor: Introduction and manual, version 5. *Faculty Publications and Other Works – EECS*. [189](#)

- [79] MacLennan, B. J. (2019). Continuum mechanics for coordinating massive microrobot swarms: Self-assembly through artificial morphogenesis. *Novel Design and Applications of Robotics Technologies*, pages 96–133. [8](#)
- [80] MacLennan, B. J. and McBride, A. C. (2020). Swarm Intelligence for Morphogenetic Engineering. In Schumann, A., editor, *Swarm Intelligence: From Social Bacteria to Humans*, pages 9–54. CRC Press. [5](#), [8](#), [80](#), [189](#)
- [81] Mamei, M., Vasirani, M., and Zambonelli, F. (2004). Experiments of morphogenesis in swarms of simple mobile robots. *Applied Artificial Intelligence*, 18(9/10):903–919. [32](#), [33](#)
- [82] Mamei, M., Zambonelli, F., and Leonardi, L. (2003). Tuples on the air: A middleware for context-aware computing in dynamic networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, pages 342–347. [33](#)
- [83] Maningo, J. M. Z., Faelden, G. E. U., Nakano, R. C. S., Bandala, A. A., Vicerra, R. R. P., and Dadios, E. P. (2016). Formation control in quadrotor swarm aggregation using Smoothed Particle Hydrodynamics. In *Region 10 Conference (TENCON), 2016 IEEE*, pages 2070–2075. IEEE. [28](#), [83](#)
- [84] Martin, D. C. (1970). Analog computers in science education. In *Computers in Undergraduate Science Education*, Chicago, Illinois. Commission on College Physics. [185](#)
- [85] Meng, Y., Guo, H., and Jin, Y. (2013). A morphogenetic approach to flexible and robust shape formation for swarm robotic systems. *Robotics and Autonomous Systems*, 61(1):25–38. [104](#), [185](#)
- [86] Meyer, M., Devesa, A., Hickel, S., Hu, X. Y., and Adams, N. A. (2010). A conservative immersed interface method for Large-Eddy Simulation of incompressible flows. *Journal of Computational Physics*, 229(18):6300–6317. [210](#)
- [87] Michelusi, N. (2017). On population density estimation via quorum sensing. In *2017 15th Canadian Workshop on Information Theory (CWIT)*, pages 1–5. [36](#)
- [88] Mijalkov, M., McDaniel, A., Wehr, J., and Volpe, G. (2016). Engineering sensorial delay to control phototaxis and emergent collective behaviors. *Physical Review X*, 6(1):011008. [36](#)
- [89] Milo, R. and Phillips, R. (2016). How fast do cells move? In *Cell Biology by the Numbers*, pages 318–321. Garland Science, Taylor & Francis Group, New York, NY. [116](#)

- [90] Miyashita, S., Guitron, S., Yoshida, K., Li, S., Damian, D. D., and Rus, D. (2016). Ingestible, controllable, and degradable origami robot for patching stomach wounds. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 909–916. [3](#)
- [91] Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574. [21](#)
- [92] Monaghan, J. J. (2005). Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703. [21](#)
- [93] Morris, J. P. (2000). Simulating surface tension with smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 33(3):333–353. [83](#)
- [94] Müller, P., Rogers, K. W., Yu, S. R., Brand, M., and Schier, A. F. (2013). Morphogen transport. *Development*, 140(8):1621–1638. [54](#), [56](#)
- [95] Müller, P. and Schier, A. F. (2011). Extracellular movement of signaling molecules. *Developmental Cell*, 21(1):145–158. [56](#)
- [96] Nagpal, R. (2001). *Programmable Self-Assembly: Constructing Global Shape Using Biologically-Inspired Local Interactions and Origami Mathematics*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts. [11](#), [15](#)
- [97] Nagpal, R. (2002). Programmable self-assembly using biologically-inspired multiagent control. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, pages 418–425. ACM. [11](#)
- [98] Nagpal, R. (2004). A catalog of biologically-inspired primitives for engineering self-organization. In Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., and Zambonelli, F., editors, *Engineering Self-Organising Systems*, Lecture Notes in Computer Science, pages 53–62. Springer Berlin Heidelberg. [12](#)
- [99] Nagpal, R. (2006). Engineering amorphous systems, using global-to-local compilation. In Braha, D., Minai, A. A., and Bar-Yam, Y., editors, *Complex Engineered Systems: Science Meets Technology*, Understanding Complex Systems, pages 291–306. Springer, Berlin. [11](#)

- [100] Nielsen, A. A. K., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016). Genetic circuit design automation. *Science*, 352(6281). 104
- [101] Occupational Safety & Health Administration (n.d.). Biological agents. <https://www.osha.gov/biological-agents>. Accessed March 15, 2022. 187
- [102] Oh, H., Ramezan Shirazi, A., Sun, C., and Jin, Y. (2017). Bio-inspired self-organising multi-robot pattern formation: A review. *Robotics and Autonomous Systems*, 91:83–100. 3
- [103] Oh, H., Shiraz, A. R., and Jin, Y. (2018). Morphogen diffusion algorithms for tracking and herding using a swarm of kilobots. *Soft Computing*, 22(6):1833–1844. 32
- [104] Paç, M. R. (2007). A fluid dynamics framework for control of mobile robot networks. Master’s thesis, Middle East Technical University. 24, 25, 26
- [105] Pac, M. R., Erkmen, A. M., and Erkmen, I. (2006a). Scalable self-deployment of mobile sensor networks: A fluid dynamics approach. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1446–1451, Beijing, China. IEEE. 24
- [106] Pac, M. R., Erkmen, A. M., and Erkmen, I. (2006b). Towards fluent sensor networks: A scalable and robust self-deployment approach. In *Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems*, pages 365–372. IEEE. 24
- [107] Pac, M. R., Erkmen, A. M., and Erkmen, I. (2007). Control of robotic swarm behaviors based on smoothed particle hydrodynamics. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4194–4200, San Diego, California. IEEE. 24, 25
- [108] Palsson, E. and Cox, E. C. (1996). Origin and evolution of circular waves and spirals in *Dictyostelium discoideum* territories. *Proceedings of the National Academy of Sciences*, 93(3):1151–1155. 56
- [109] Pascalie, J., Potier, M., Kowaliw, T., Giavitto, J.-L., Michel, O., Spicher, A., and Doursat, R. (2016). Developmental design of synthetic bacterial architectures by morphogenetic engineering. *ACS Synthetic Biology*, 5(8):842–861. 2, 15, 31, 33

- [110] Patra, M. and Karttunen, M. (2006). Stencils with isotropic discretization error for differential operators. *Numerical Methods for Partial Differential Equations*, 22(4):936–953. [122](#)
- [111] Peng, L. and Mohseni, K. (2014). Sensor driven feedback for puff estimation using unmanned aerial vehicles. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 562–569, Orlando, Florida. IEEE. [27](#)
- [112] Perkinson, J. R. and Shafai, B. (2005). A decentralized control algorithm for scalable robotic swarms based on mesh-free particle hydrodynamics. In *Proceedings of the IASTED International Conference on Robotics and Applications*, pages 1–6, Cambridge, Massachusetts. [4](#), [22](#), [24](#), [25](#)
- [113] Petrů, L. and Wiedermann, J. (2014). A robust universal flying amorphous computer. In Calude, C. S., Freivalds, R., and Kazuo, I., editors, *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday*, Lecture Notes in Computer Science, pages 421–435. Springer International Publishing, Cham. [13](#)
- [114] Picker, A., Cavodeassi, F., Machate, A., Bernauer, S., Hans, S., Abe, G., Kawakami, K., Wilson, S. W., and Brand, M. (2009). Dynamic coupling of pattern formation and morphogenesis in the developing vertebrate retina. *PLOS Biology*, 7(10):e1000214. [1](#)
- [115] Pimenta, L. C., Michael, N., Mesquita, R. C., Pereira, G. A., and Kumar, V. (2008). Control of swarms based on hydrodynamic models. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference On*, pages 1948–1953. IEEE. [27](#)
- [116] Pimenta, L. C. A., Mendes, M. L., Mesquita, R. C., and Pereira, G. A. S. (2007). Fluids in electrostatic fields: An analogy for multirobot control. *IEEE Transactions on Magnetics*, 43(4):1765–1768. [27](#)
- [117] Pimenta, L. C. A., Pereira, G. A. S., Michael, N., Mesquita, R. C., Bosque, M. M., Chaimowicz, L., and Kumar, V. (2013). Swarm coordination based on smoothed particle hydrodynamics technique. *IEEE Transactions on Robotics*, 29(2):383–399. [27](#)
- [118] Polynikis, A., Hogan, S. J., and di Bernardo, M. (2009). Comparing different ODE modelling approaches for gene regulatory networks. *Journal of Theoretical Biology*, 261(4):511–530. [31](#), [104](#)
- [119] Powers, S. (2016). Asset protection in escorting using multi-robot systems. Master’s thesis, Rochester Institute of Technology, Rochester, New York. [29](#)

- [120] Price, J. F. (2006). *Lagrangian and Eulerian Representations of Fluid Flow: Kinematics and the Equations of Motion*. Woods Hole Oceanographic Institution. [9](#)
- [121] Purcell, E. M. (1977). Life at low Reynolds number. *American journal of physics*, 45(1):3–11. [116](#), [121](#), [123](#)
- [122] Qi, J., Vazquez, R., and Krstic, M. (2015). Multi-agent deployment in 3-D via PDE control. *IEEE Transactions on Automatic Control*, 60(4):891–906. [15](#)
- [123] Qiu, T., Lee, T.-C., Mark, A. G., Morozov, K. I., Münster, R., Mierka, O., Turek, S., Leshansky, A. M., and Fischer, P. (2014). Swimming by reciprocal motion at low Reynolds number. *Nature Communications*, 5:5119. [121](#)
- [124] Ramachandran, R. K., Elamvazhuthi, K., and Berman, S. (2018). An optimal control approach to mapping GPS-denied environments using a stochastic robotic swarm. In *Robotics Research*, pages 477–493. Springer. [16](#)
- [125] Ramezan Shirazi, A., Oh, H., and Jin, Y. (2014). Morphogenetic self-organization of collective movement without directional sensing. In Mistry, M., Leonardis, A., Witkowski, M., and Melhuish, C., editors, *Advances in Autonomous Robotics Systems*, volume 8717, pages 139–150. Springer International Publishing, Cham. [33](#)
- [126] Ream, R. A., Theriot, J. A., and Somero, G. N. (2003). Influences of thermal acclimation and acute temperature change on the motility of epithelial wound-healing cells (keratocytes) of tropical, temperate and Antarctic fish. *Journal of Experimental Biology*, 206(24):4539–4551. [116](#)
- [127] Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., and Nagpal, R. (2014a). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975. [30](#), [185](#)
- [128] Rubenstein, M., Cornejo, A., and Nagpal, R. (2014b). Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799. [32](#)
- [129] Şahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. In *International Workshop on Swarm Robotics*, pages 10–20. Springer. [14](#)
- [130] Sayama, H. (2011). Seeking open-ended evolution in swarm chemistry. In *2011 IEEE Symposium on Artificial Life (ALIFE)*, pages 186–193. [6](#)

- [131] Schmidt, C. K., Medina-Sánchez, M., Edmondson, R. J., and Schmidt, O. G. (2020). Engineering microrobots for targeted cancer therapies from a medical perspective. *Nature Communications*, 11(1):5618. [3](#)
- [132] Sevenler, D., Trueb, J., and Ünlü, M. S. (2019). Beating the reaction limits of biosensor sensitivity with dynamic tracking of single binding events. *Proceedings of the National Academy of Sciences*, 116(10):4129–4134. [113](#)
- [133] Shaw, A. and Mohseni, K. (2010). Communication characterization for a ZigBee based airborne wireless sensor network. In *AIAA Guidance, Navigation, and Control Conference*, Toronto, Canada. American Institute of Aeronautics and Astronautics. [27](#)
- [134] Shaw, A. and Mohseni, K. (2011). A Fluid Dynamic Based Coordination of a Wireless Sensor Network of Unmanned Aerial Vehicles: 3-D Simulation and Wireless Communication Characterization. *IEEE Sensors Journal*, 11(3):722–736. [27](#)
- [135] Shimizu, M., Ishiguro, A., Kawakatsu, T., Masubuchi, Y., and Doi, M. (2003). Coherent swarming from local interaction by exploiting molecular dynamics and stokesian dynamics methods. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, volume 2, pages 1614–1619. IEEE. [23](#)
- [136] Silic, M. B., Song, Z., and Mohseni, K. (2018). Anisotropic flocking control of distributed multi-agent systems using fluid abstraction. In *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, Kissimmee, Florida. American Institute of Aeronautics and Astronautics. [27](#), [28](#)
- [137] Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., and Sharpe, J. (2018). Morphogenesis in robot swarms. *Science Robotics*, 3(25). [16](#), [32](#), [104](#)
- [138] Somogyi, E. (2018). A dynamic non-manifold mesh data structure to represent biological materials. *Journal of WSCG*, 26:21–30. [10](#)
- [139] Somogyi, E. and Glazier, J. A. (2017). A modeling and simulation language for biological cells with coupled mechanical and chemical processes. In *TMS/DEVS Symposium on Theory of Modeling & Simulation (TMS/DEVS 2017)*, Virginia Beach, VA, USA. Society for Modeling and Simulation International (SCS). [10](#)

- [140] Song, Z., Lipinski, D., and Mohseni, K. (2017). Multi-vehicle cooperation and nearly fuel-optimal flock guidance in strong background flows. *Ocean Engineering*, 141:388–404. [27](#), [28](#), [118](#)
- [141] Spears, W. M. and Spears, D. F., editors (2011). *Physicomimetics: Physics-Based Swarm Intelligence*. Springer, Heidelberg. [23](#), [24](#)
- [142] Spears, W. M., Spears, D. F., Heil, R., Kerr, W., and Hettiarachchi, S. (2005). An overview of physicomimetics. In Şahin, E. and Spears, W. M., editors, *Swarm Robotics*, Lecture Notes in Computer Science, pages 84–97, Berlin, Heidelberg. Springer. [23](#)
- [143] Teague, B. P., Guye, P., and Weiss, R. (2016). Synthetic morphogenesis. *Cold Spring Harbor Perspectives in Biology*, 8(9). [3](#), [31](#)
- [144] Temel, F. Z. and Yesilyurt, S. (2013). Simulation-based analysis of micro-robots swimming at the center and near the wall of circular mini-channels. *Microfluidics and Nanofluidics*, 14(1-2):287–298. [121](#), [123](#)
- [145] Thomas, P. J. and Eckford, A. W. (2016). Shannon capacity of signal transduction for multiple independent receptors. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1804–1808. [113](#)
- [146] Tilki, U. and Erkmen, A. M. (2015). Fluid swarm formation control for hand gesture imitation by ellipse fitting. *Electronics Letters*, 51(6):473–475. [26](#)
- [147] Tilki, U., Erkmen, I., and Erkmen, A. M. (2010). Imitation of hand postures with particle based fluidics method. In *Int. Conf. on Intelligent Automation and Robotics*. [26](#)
- [148] Tilki, U., Erkmen, I., and Erkmen, A. M. (2011). Imitation of basic hand preshapes by fluid based method: Fluidics formation control. *Turkish Journal of Electrical Engineering & Computer Sciences*, 19(2):221–234. [26](#)
- [149] Tilki, U., Erkmen, I., and Erkmen, A. M. (2012a). Human hand gesture imitation by the formation control of a fluid swarm modeled by smoothed particle hydrodynamics (sph). 2(6). [26](#)
- [150] Tilki, U., Erkmen, I., and Erkmen, A. M. (2012b). Imitation of human body poses by the formation control of a fluidic swarm. pages 361–370. ASME. [26](#)

- [151] Tseng, Y.-H. and Ferziger, J. H. (2003). A ghost-cell immersed boundary method for flow in complex geometry. *Journal of Computational Physics*, 192(2):593–623. [210](#)
- [152] Uhlmann, M. (2005). An immersed boundary method with direct forcing for the simulation of particulate flows. *Journal of Computational Physics*, 209(2):448–476. [210](#)
- [153] Valentini, G., Hamann, H., and Dorigo, M. (2016). Global-to-local design for self-organized task allocation in swarms. Technical report, Technical report TR/IRIDIA/2016-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. [18](#)
- [154] Vallée-Bélisle, A., Ricci, F., and Plaxco, K. W. (2012). Engineering biosensors with extended, narrowed, or arbitrarily edited dynamic range. *Journal of the American Chemical Society*, 134(6):2876–2879. [57](#), [113](#)
- [155] Van der Klauw, M. C. R. (2015). Mobile robot swarming using radio signal strength measurements and dead-reckoning. Master’s thesis, Delft University of Technology. [36](#)
- [156] Verbruggen, M. (2019). The question of swarms control: Challenges to ensuring human control over military swarms. *Non-Proliferation and Disarmament Papers*, (65). [187](#)
- [157] Vicsek, T. and Zafeiris, A. (2012). Collective motion. *Physics Reports*, 517(3):71–140. [65](#)
- [158] Wang, F. and Zhang, W. (2019). Synthetic biology: Recent progress, biosafety and biosecurity concerns, and possible solutions. *Journal of Biosafety and Biosecurity*, 1(1):22–30. [188](#)
- [159] Wasilow, S. and Thorpe, J. B. (2019). Artificial intelligence, robotics, ethics, and the military: A Canadian perspective. *AI Magazine*, 40(1):37–48. [187](#)
- [160] Werfel, J., Petersen, K., and Nagpal, R. (2014). Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758. [17](#)
- [161] Wiedermann, J. (2011). Nanomachine computing by quorum sensing. In Kelemen, J. and Kelemenová, A., editors, *Computation, Cooperation, and Life: Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 203–215. Springer Berlin Heidelberg, Berlin, Heidelberg. [35](#), [36](#)
- [162] Wiedermann, J. and Petrů, L. (2009). Communicating mobile nano-machines and their computational power. In Cheng, M., editor, *Nano-Net*, Lecture Notes of the Institute for

Computer Sciences, Social Informatics and Telecommunications Engineering, pages 123–130. Springer Berlin Heidelberg. [34](#)

- [163] Yamins, D. (2007). *A Theory of Local-to-Global Algorithms for One-Dimensional Spatial Multi-Agent Systems*. PhD thesis, Harvard University, Cambridge, Massachusetts. [11](#), [17](#)
- [164] Yamins, D. and Nagpal, R. (2008). Automated global-to-local programming in 1-D spatial multi-agent systems. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 615–622. International Foundation for Autonomous Agents and Multiagent Systems. [17](#)
- [165] Yang, C. and Zhang, H. (2019). Density-based pattern control for spacecraft swarms around circular orbits. In *2019 Chinese Control Conference (CCC)*, pages 5915–5921. [24](#)
- [166] Zacharia, I. G. and Deen, W. M. (2005). Diffusivity and solubility of nitric oxide in water and saline. *Annals of Biomedical Engineering*, 33(2):214–222. [54](#)
- [167] Zhao, S. (2010). *Multi-Robot Cooperative Control: From Theory to Practice*. PhD thesis, University of Cincinnati, Cincinnati, Ohio. [29](#)
- [168] Zhao, S., Ramakrishnan, S., and Kumar, M. (2011). Density-based control of multiple robots. In *American Control Conference (ACC), 2011*, pages 481–486. IEEE. [29](#)

Appendices

A Computational Fluid Dynamics

A computational fluid dynamics (CFD) method suitable for this model must capture three key details. First, it must handle a large number of independently moving immersed solid boundaries. Second, it must model diffusion of dilute solutes (sometimes called “scalar transport” in CFD literature). Finally, it must be stable and meaningful even close to immersed boundaries, because it is at these boundaries that agents must measure solute concentration.

At the same time, I prefer a method that is computationally efficient and easy to code. It would be premature to sacrifice too much time for improved accuracy at this stage of research. I wish to explore whether NSF-SPH is qualitatively valid in the presence of fluid dynamics; answering more specific practical questions would require both three-dimensional modeling and propulsion details.

I did not find a suitable off-the-shelf CFD solution. Existing frameworks and libraries that allow for moving immersed boundaries are intended for what one might call off-line use cases: The user specifies a physical scenario (for example, a propeller turning in a tunnel), the simulation runs for a given amount of time, and the results are analyzed. By contrast, my simulation must do three things between every fluid dynamics timestep: sense morphogen concentrations at variable positions, update agents’ poses and velocities, and inject morphogen at variable positions. While not impossible to use, say, the OpenFOAM framework in this manner, doing so would require additional development and simulation time.

A.1 Modeling Fluid Velocity

Lacking a suitable off-the-shelf CFD method, I implement a basic model of fluid velocity myself. I considered a variety of methods from the literature. Major categories of CFD methods include finite difference methods (FDM), finite element methods, and particle methods. All are able to model moving immersed solid boundaries. I chose to use an FDM; FDMs are appealing because of their simplicity, both in terms of computation and development. They work well with a fixed and regular rectangular grid. Finite element methods are more complex and require frequent re-meshing when solid boundaries are moved. Particle methods, such as SPH or the Particle-in-Cell family of methods, have clearer advantages when fluid density or fluid interfaces are of interest. By contrast, for the present scenario we are interested in flow within an incompressible fluid. However, one particle method, lattice Boltzmann, may be worth exploring for future simulation, because Liu and Wu [69] have developed a method for modeling immersed particles that may be suitable.

In the context of FDMs, various methods exist for modeling moving immersed solid boundaries, for example, cut-cell methods, ghost-cell methods, and reconstruction methods (e.g., Meyer et al. [86], Tseng and Ferziger [151], and Kang et al. [51], respectively). Each method addresses the challenge of enforcing Dirichlet boundary conditions for velocity around an immersed body, even though the body’s boundary does not generally coincide with grid cell boundaries. These boundary conditions are generally “no-slip,” or homogeneous relative to the body’s motion, implying that the fluid immediately next to an immersed body travels along with that body. Cut-cell methods involve cutting the grid cells that an immersed body’s boundary passes through into purely solid and purely fluid sub-cells with triangular or trapezoidal shapes. Making these sub-cells allows no-slip conditions to be enforced at grid cell boundaries, but the non-rectangular grid cells require methods similar to finite-volume methods. Ghost-cell and reconstruction methods enforce velocities associated with grid cells in the interior of the immersed body to be faster than the body’s motion, such that a velocity interpolation on the body’s boundary itself would be as desired. As with cut-cell methods, these methods require additional geometrical calculations at each timestep relating to the precise location and angle of an immersed boundary.

I base my fluid velocity simulation on a method from Kajishima et al. [50] that is known for simplicity and efficiency [152]. It simply imposes the body’s velocity on all the grid cells completely within the body as one step of the time-marching loop. For grid cells that the body’s boundary cuts through, it assigns a weighted average of the body velocity and existing fluid velocity, based on the proportion of the grid cell that is fluid as opposed to solid (which I’ll call the “fluid fraction”).

As with many FDM methods in the context of incompressible flows, the method of Kajishima et al. [50] is a projection method. Projection methods address viscous and pressure forces in separate steps [16]. The simplest form of this method updates the velocity field in two steps. First, it computes velocity change due to viscosity (that is, diffusion of momentum) and inertia, ignoring pressure. This step results in an intermediate velocity field $\mathbf{u}^*$ with nonzero divergence. Second, a Poisson equation is solved for ψ , a pseudo-pressure field:

$$\nabla^2\psi = -\nabla \cdot \mathbf{u}^*.$$

(Physical pressure is $\rho\psi/\Delta t$ where ρ is fluid density, but this detail can be ignored here as it is not needed for other calculations.) An update of the velocity field is completed by subtracting the

gradient of ψ from $\mathbf{u}^*$. This procedure projects the intermediate field onto the space of divergence-free velocity fields, taking the solenoidal part of a Helmholtz decomposition. The result approximates the simultaneous solution of the full Navier-Stokes momentum equation.

The method of Kajishima et al. [50] is simple as it is, but the parts relevant here are simpler still, because we ignore inertial calculations as well as fluid forces acting on agents. In essence, the time-marching loop becomes:

1. Diffuse fluid momentum (to account for viscous forces)
2. Compute pseudo-pressure field
3. Subtract the gradient of pseudo-pressure from velocity
4. Impose body (agent) velocities, weighted by fluid fraction

I make two changes to the order of these steps. First, I actually use the velocity field (that is, compute morphogen advection) between the third and fourth steps, instead of after the fourth. Second, I impose agent velocities an additional time, after momentum diffusion.

Computing advection directly after subtracting the pseudo-pressure gradient is an important trade-off. At this point in the loop, the velocity field is approximately divergence free, but the velocity field at the boundaries of agents and in their interior no longer matches agents' velocities, a violation of the no-slip condition. Specifically, the velocity field in agents' interiors will be noticeably slower than the agents' true velocities, as though fluid can pass through the agents to a slight extent. By contrast, after re-imposing agent body velocities, the velocity field, while still mostly divergence-free, has nonzero divergence most notably in the immediate vicinity of agent bodies' boundaries. This divergence leads to a slight buildup of morphogen in front of an agent and a decrease behind, as well as various less predictable artifacts, especially as an agent is turning. These artifacts might be of less concern if I was studying overall fluid behavior, but because of the nature of NSF-SPH, agents rely on measuring morphogen concentrations at their own boundary. In reality, morphogen concentrations would be smooth and would not build up in front of an agent, and capturing these qualitative traits is more important for modeling NSF-SPH than enforcing the no-slip condition exactly.

Kajishima et al. [50] ignore the agent body during the computation of momentum diffusion due to viscosity. However, in the context of our low-Reynolds-number environment, timesteps large enough to be practical lead to a noticeable amount of momentum lost from the agent bodies. To

help make up for this loss, I reimpose the body velocity once more after momentum diffusion. In principle, momentum should be diffused with the momentum of the body itself fixed by boundary conditions. But such boundary conditions require changing the Laplacian matrix at each timestep. I tested this informally, weighting matrix entries by fluid fraction, and found that it slowed down the simulation with no visually discernible change in the velocity field. Studying this issue further could be fruitful.

With the above modifications, my simplification of the method of Kajishima et al. [50] becomes:

1. Impose body velocities, weighted by fluid fraction
2. Diffuse fluid momentum
3. Impose body velocities once again
4. Compute pseudo-pressure field
5. Subtract the gradient of pseudo-pressure from velocity

I make one further simplification to the basic method of Kajishima et al. [50]. Because it uses a staggered grid, fluid fraction information must be centered on grid edges, not cell centers, where I calculate fluid fractions for other purposes in the simulation such as visualization and diffusion. Rather than compute two additional, offset grids of fluid fractions for horizontal and vertical edges, I simply use information I already compute to find the proportion of each one-dimensional edge that is not covered by agents' bodies. An alternative would be to average together neighboring fluid fractions, but from informal testing, this averaging spreads out boundary locations more than is desirable. In the future, it is likely that computing additional staggered grids for fluid fraction could decrease grid dependence without sacrificing too much speed.

My solving methods are also different in their particulars than those of Kajishima et al. [50], though I don't consider these differences to deviate from the essence of their method. I choose solving methods that are easy to implement or for which I can easily call open source library routines. I use first- and second-order central differences for first and second derivatives, instead of fourth-order schemes. I use the backward Euler method for momentum diffusion, instead of Adams-Bashforth. I solve the relevant matrix equations for momentum diffusion and pseudo-pressure using the biconjugate gradient stabilized method from the Eigen library [39].

This method seems sufficient to subjectively capture the effect of agents' translational motion on surrounding fluid velocity. Figure 1 shows the result of an agent 40 μm in diameter moving 50 μm

rightward, one quarter of the way across a $200\text{ }\mu\text{m} \times 200\text{ }\mu\text{m}$ square, at a typical speed $0.1\text{ }\mu\text{m s}^{-1}$. The blue squares are attached to the liquid medium to show its movement. The simulation timestep is one second, also typical for my simulations. Figure 1a has resolution such that each agent is no fewer than six grid cells across, which I choose as a minimum resolution for simulations that include fluid flow (i.e., swimming agents). (My overall grids are, of course, contain many agents and are much larger.) Figure 1b has the same timestep of one second but a far higher spatial resolution; I do not use such high resolution relative to agent bodies in my simulations, but I include this subfigure to provide a clearer subjective illustration of my fluid velocity model.

Ideally, a simulation would also capture the subjective effect of agents' rotational motion on surrounding fluid velocity. The method I am using is capable of doing so. Unfortunately, at the very high Reynolds number scales involved, subjectively adequate coupling between viscosity and pressure requires timesteps 0.01 s or smaller. Such small timesteps with the computationally intensive fluid velocity solver are infeasible for my work. I do not expect this limitation to significantly impact my results, because the agents turn slowly (section 4.7). If it proves important in the future to resolve the effects of agent rotation, it may be worthwhile to explore a CFD method that, unlike projection methods, solves for both viscosity and pressure simultaneously.

A.2 Modeling Diffusion

Using FDM for fluid advection makes it simple to re-use the same method described in section 4.11.1 to model diffusion, except that we now need to apply homogeneous Neumann boundary conditions at the edges of agents' bodies. As with fluid advection, I want to rely on the proportion of each cell occupied, and not on the precise geometry of the agent body within the grid cell. As in section 4.11.1, consider diffusion in terms of transfers between neighboring grid cells. When boundaries follow grid cell borders, we can apply Neumann boundary conditions by simply not transferring between two cells if one of them is obstructed. Similarly, when a boundary cuts through the interior of a grid cell, we can transfer part of what would otherwise be transferred, according to the proportion of unobstructed space in the partially obstructed grid cell. This method can be seen as a linear interpolation between the cases where the obstructed cell were completely free versus fully obstructed. If both cells are partly obstructed, we transfer an amount according to the minimum unobstructed proportion between the two cells. This linear interpolation can also be given a loose geometric interpretation. Assume the unobstructed proportion of the boundary between two grid cells is the minimum unobstructed proportion between the two cells, as would be the case if the two

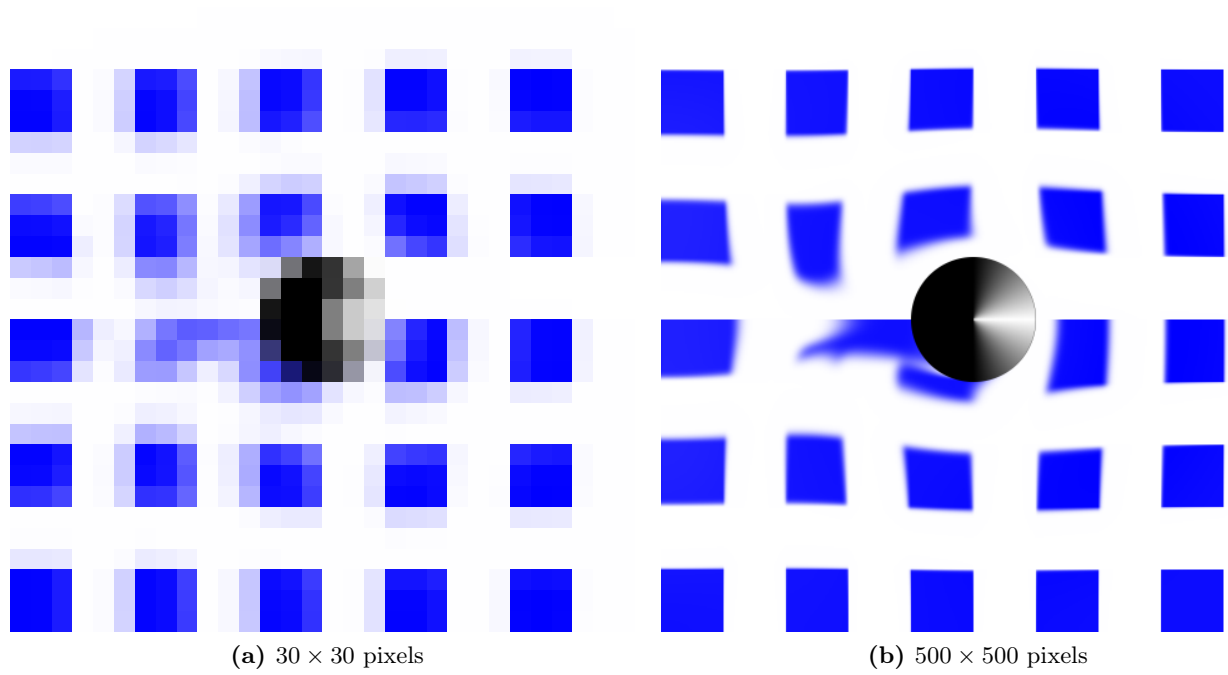


Figure 1: Motion of fluid in response to agent translation, for two different spatial resolutions. The agent, $40\text{ }\mu\text{m}$ in diameter, has traveled rightward to the center of the $200\text{ }\mu\text{m} \times 200\text{ }\mu\text{m}$ domain from a point halfway between the center and the left wall. The agent's speed is $0.1\text{ }\mu\text{m s}^{-1}$, and the timestep is one second. Distortion of what begins as a regular grid of blue squares illustrates displacement of the fluid medium.

cells are adjacent and the surface of the obstruction cutting through the cells is perpendicular to the boundary between them (fig. 2). The total transfer of morphogen from one cell to the other is the integral of diffusive flux over the boundary between them, and thus proportional to the length of this boundary, which is the same proportion as the minimum unobstructed area. This geometry has the virtue that the diffusion transfer goes to zero as unobstructed area in one of the cells goes to zero. Improving on these relatively crude boundary conditions would seem to require a more detailed accounting of the actual geometry of the boundary.

A.3 Stalks for Agents' Instruments

I next describe a challenge implied by the above approach and solved by locating agents' instruments on short stalks. Tracking morphogen concentration, rather than mass, in our Eulerian grid is the natural choice for advection. Because the advection equation relates velocity to concentration rather than mass, tracking mass would require calculating concentration by dividing mass by the fluid fraction of each partially obstructed grid cell. However, when the upstream and downstream grid cells for a given advection transfer have different fluid fractions, there is no set of choices regarding which cells' fluid fractions to use in the calculation such that both mass conservation and divergence-free flow are respected. (I tested various such options informally, and they resulted in prominent artifacts near agent bodies.)

Unfortunately, tracking concentration instead of mass presents a challenge for modeling the morphogen-production process. We want morphogen to be produced at particular rates of mass per unit time. But if a producing instrument's location is within a partially obstructed grid cell, how should production affect that grid cell's concentration?

One option would be to take the fluid fraction of the grid cell into account and increase concentration by the correct amount for a given incremental mass addition. But because advection does not consider fluid fraction during the application of a velocity field to a concentration field, mass will not be conserved during advection. For example, consider a producing instrument in a tiny fluid corner of a mostly-obstructed grid cell. The morphogen concentration in this cell would be increased a great deal by injection of even a small mass of morphogen. But when the velocity field is applied, this high concentration will typically be advected to neighboring cells with larger fluid fractions. This result is incoherent: the concentration change in the neighboring grid cell should not depend on the fluid fraction in the first cell.

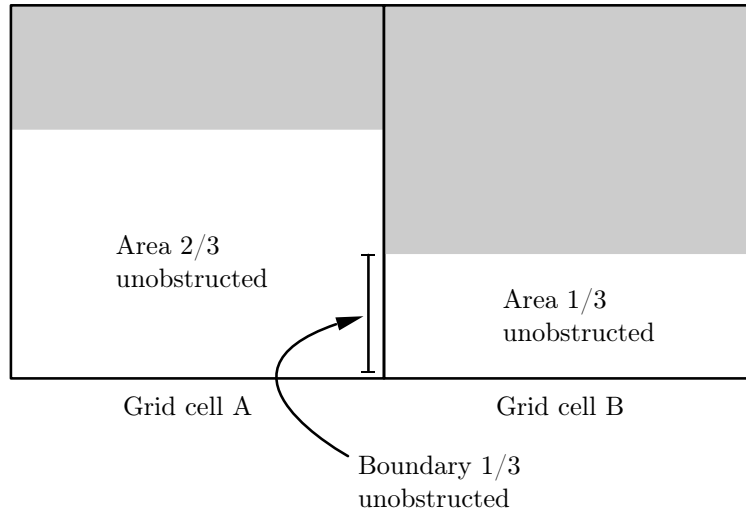


Figure 2: Assumed geometry of partially obstructed cells for diffusion boundary conditions. I assume the agent boundary is stair-step shaped as shown. The agent boundary follows grid cell boundaries before cutting through grid cells perpendicular to those cell boundaries. Under this assumption, the proportional length of unobstructed cell boundary is the same as the proportion of unobstructed area in the more-obstructed grid cell. The amount of morphogen transferred between the two grid cells is multiplied by this proportion.

Another option would be to ignore fluid fraction in the grid cell containing a production device, increasing concentration in the grid cell by the same amount regardless. This option maintains mass conservation, but invalidates diffusion. Consider again the above example in which a small corner of a grid cell is fluid and contains a production device. Diffusion from this cell to neighboring cells with larger fluid fractions will be smaller than if the first cell had a larger fluid fraction, but it ought to be similar, because the smaller fraction should be offset by a larger concentration gradient.

I partially solve this dilemma by simply moving the production devices away from the agent bodies the distance of one grid cell's diagonal, as though they are placed on short stalks around the agent. I move sensing devices similarly, for simplicity. The problem I am addressing here is strictly numerical and should not reflect any need for stalks on physical agents. Also, the problem is not solved entirely: Agents' production devices can still find themselves in grid cells partially occupied by another agent's body. In this case, I take the second horn of the dilemma. That is, I model morphogen production as increasing the morphogen concentration in the production device's grid cell without regard for that cell's fluid fraction. This choice prioritizes mass conservation over correct diffusion rate. In a typical Morphgen application, the two agents will move apart after a short time, relieving the problem.

Vita

Allen McBride holds a Bachelor of Arts degree in biology from Swarthmore College, a Master of Arts degree in ecology from Duke University, and a Master of Science degree in computer science from the University of Tennessee. He has worked as Post-Master's Research Associate in the Environmental Sciences Division at Oak Ridge National Laboratory. He is originally from Oak Ridge, Tennessee.