A Morphogenetic Program for Path Formation by Continuous Flocking

BRUCE J. MACLENNAN*

Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, USA

Received 11 June 2018

Artificial morphogenesis uses processes inspired by embryology to control massive swarms of microscopic agents to assemble complex physical structures, but this requires new means for describing these processes. Here we use an example morphogenetic program to illustrate a prototype implementation of *morphgen*, a morphogenetic programming language. The syntax and semantics are described informally and illustrated by the example program, which is included in its entirety in an appendix. Another appendix includes a complete formal grammar for the current version of the language. Next, we describe the results of a series of experiments with the program, which simulates a continuous swarm of microscopic agents creating paths from an origin to a destination while avoiding obstacles. We present the effects of various parameters and of alternative ways of accomplishing particular purposes.

Key words: artificial morphogenesis, continuous flocking, morphgen, morphogenetic engineering, swarm robotics

1 INTRODUCTION

Artificial morphogenesis uses processes inspired by embryology to control massive swarms of robots to assemble complex physical structures. It is

* email: maclennan@utk.edu

inspired by the observation that in embryological development trillions of microscopic cells self-organize and coordinate with each other to assemble complex, hierarchically structured three-dimensional forms; it is a variety of *morphogenetic engineering* [6, 20, 21]. Our approach uses partial differential equations (PDEs) to describe massive swarms of microscopic agents (e.g., microscopic robots or genetically engineered motile cells). PDEs are the language preferred by embryologists for describing morphogenesis [2, 5, 19, 24], but PDEs also extrapolate the number and smallness of the agents to the continuum limit, which leads to algorithms that scale to very large numbers of very small agents. Artificial morphogenesis is one approach toward general programmable matter [15] and has some similarities to *amorphous computing* [1], at least in the early stages of morphogenesis when the mass may be relatively unstructured.

In this article we use a simple example of artificial morphogenesis and massive swarm robotics to illustrate a morphogenetic programming language. The concept of artificial morphogenesis and the morphogenetic programming notation on which this language is based are described in more detail in a number of previous papers and reports [7, 8, 9, 11, 12, 13, 14, 15, 16].

The purpose of our example morphogenetic program is to lay down path material from a starting location to a destination while avoiding collisions with already created paths. A typical application would be routing dense bundles of nerve-like fibers between regions of an artificial brain [10]. The path is laid down by a massive swarm of microscopic robots following a chemical attractant diffusing from the destination. Bird flocking and fish schooling are familiar self-organizing processes (e.g., [22, 23]), and there has been some research into *continuous flocking*, which treats the agent swarm as a continuum [3, 4, 25]. We have used a modified flocking algorithm to control simulated agents creating paths between designated origins and positions while avoiding collisions with existing paths [13, 14, 16]. Here we use two-dimensional continuous flocking to create paths from an origin to a destination while avoiding obstacles.

The morphogenetic programming language, tentatively named *morphgen*, adheres fairly closely to the mathematical notation used in publications. The prototype translator illustrated in this report is implemented by a *syntax macroprocessor* (tentatively named "synmac") [18]. Like more familiar macroprocessors, it uses a set of macro definitions to translate a source language into a target language. In this case, the source language is morphgen and the target language is MATLAB® or compatible GNU Octave. Although synmac is quite flexible, it does not include a full parser, and so some syntactic con-

cessions must be made in this prototype implementation. They will be mentioned in the appropriate places below. There are two very similar dialects of morphgen, *morphgen2D* for two-dimensional simulations and *morphgen3D* for three-dimensional simulations. The grammar for the current version of morphgen2D is given in Appendix B.

2 DESCRIPTION OF THE MORPHOGENETIC PROGRAM

The complete morphogenetic program is shown in Appendix A. Much of the program is relatively self-explanatory, at least in the context of the artificial morphogenetic programming notation described in previous publications. A few particular features will be explained here.

The program begins with a specification of the simulation parameters (lines 8–13 in Appendix A):

```
simulation parameters:
  space: −1 < x < 1, −1 < y < 1
  duration = 6.75
  spatial resolution = 0.01
  temporal resolution = 0.001
end
```

The **space** specification defines the 2D space in which the simulation takes place, and the **duration** specification defines its length (both specifications in arbitrary units). The final two lines define the spatial and temporal resolutions of the simulation, which are easily changed. Therefore, in this case the spatial mesh is 200×200 and the simulation runs for 6750 iterations.

After the simulation specification comes the morphogenetic program proper. In this case it begins with the definition of four *substances*, a morphogen, the path material, the swarm, and the goal material. In the morphogenetic programming language, a substance is somewhat analogous to a class in an object-oriented programming language in that it defines a set of things with related properties and behaviors. Substances can be instantiated in particular *bodies* (analogous to objects in an object-oriented language). The definition of the *morphogen* substance (lines 15–21) is perhaps most illustrative:

```
substance morphogen:
    scalar field A
  behavior:
    param d_A = 0.03    /* diffusion constant */
```

```
    param tau_A = 100  /* decay time constant */
    D A += d_A * del^2 A - A/tau_A
end
```

The first line after the header declares that the substance is characterized by a scalar field A, which represents the concentration of the attractant morphogen at every location is the two-dimensional space. The first two lines after **behavior** simply define constants. The last line, which begins with **D**, is an approximation to the usual morphogenetic programming notation, which we have used in previous publications:

$$\text{Đ}A = d_A \nabla^2 A - A/\tau_A + \cdots. \tag{1}$$

This is an example of a *change equation*, which can be interpreted ambiguously as a partial differential equation or a finite difference equation. The lefthand side ĐA represents either a temporal partial derivative ($\partial_t A$, $\dot{A}$) or a temporal finite difference ($\Delta A/\Delta t$). The first term on the righthand side, $d_A \nabla^2 A$, describes the diffusion of the morphogen A. The second term, $-A/\tau_A$, describes decay of the morphogen so that it doesn't saturate the space. The notation "$+\cdots$" in Eq. 1 implies the other substances may extend this equations with additional terms (e.g., sources and sinks).

The equation is expressed in the morphgen programming language as follows:

```
  D A += d_A * del^2 A - A/tau_A
```

The notational change is due primarily to the syntactic limitations of the syntax macroprocessor and the fact that it is translating into MATLAB/Octave. Addition and subtraction of scalar and vector fields can be written normally, as in the above example. Multiplication and division of fields by scalars can be written with the multiplication and division operators ("$*$" and "/"); for example "A/tau_A" in the above example. The "+=" operator indicates that this is a *partial* change equation, and that other terms may be added in other substances (indicated by "$+\cdots$" in Eq. 1).

In the absence of obstacles, the steady-state concentration of morphogen at a distance r from the goal, which produces it at a rate k_G, is given by

$$A(r) = k_G \exp\left(-\frac{r}{\sqrt{d_A \tau_A}}\right)$$

that is, the characteristic length constant is $\sqrt{d_A \tau_A}$.

The goal material, which is static ($\text{Đ}G = 0$), has a slightly more complicated definition (lines 54–60):

```
substance goal_material:
    scalar field G
  behavior:
    param k_G = 100          /*attract. release rate*/
    D G = 0                  /*G field is fixed*/
    D A += k_G*[G*(1-A)]     /*goal emits attract.*/
end
```

The last line of the behavior definition is an *extension* to the definition of $\text{Đ}A$ in the definition of the morphogen substance. It adds to Eq. 1 an additional source term $k_G G(1-A)$, which describes production of morphogen A in the goal region (where $G = 1$) up to saturation ($A = 1$). Therefore the attractant diffuses continuously from the goal throughout the space.

Equation 1 in the definition of the morphogen subtance represents the inherent physical properties of the attractant and the medium through which it is diffusing, including the diffusion and decay constants. In contrast, the partial equation $\text{Đ}A \mathrel{+}= k_G G(1-A)$ in the definition of goal_material represents the controlled production of attractant by the goal agents. Partial equations are often used to separate the physical properties of a substance from effects that are controlled by the agents, as in this example.

This attractant emission equation illustrates another restriction of the prototype implementation: multiplication of two scalar fields or of a scalar field by a vector field must be surrounded by square brackets; for example, "[G*(1−A)]" is a product of scalar fields. Similarly the quotient of a vector field by a scalar field and powers of scalar fields must be surrounded by brackets.

The definition of the path substance (lines 23–29), represents material being laid down by the swarm as well any previously created paths, which are obstacles to be avoided:

```
substance path_material:
    scalar field P
  behavior:
    param tau_P = 0.2    /*absorption time const.*/
    D P += 0             /*passive path material*/
    D A -= [P*A]/tau_P   /*path absorbs attractant*/
end
```

The concentration of path material is represented by the scalar P field, which

does not change on its own, as indicated by $\text{Đ}P \mathrel{+}= 0$. The partial equation $\text{Đ}A \mathrel{-}= PA/\tau_P$ describes rapid decay or absorption of the morphogen where there is path material ($P > 0$); in effect, existing path material sucks up attractant, which steers the swarm away from these obstacles. The three partial equations together define the dynamics of the A field:

$$\text{Đ}A = d_A \nabla^2 A - A/\tau_A + k_G G(1 - A) - PA/\tau_P. \tag{2}$$

The swarm substance (lines 31–52) is the most complicated, for it describes how a continuous mass of agents follows the morphogen gradient. The first part of the definition declares two scalar fields and two vector fields:

```
substance swarm:
    scalar fields:
      C  /* swarm concentration */
      S  /* magnitude of morphogen gradient */
    end
    vector fields:
      U  /* morphogen gradient */
      V  /* swarm velocity */
    end
```

The C field, which represents the concentration of agents, is the principal field in the morphogenetic process since the agents lay down the path material, as just explained. The **behavior** part of the substance definition begins with five parameter definitions:

```
behavior:
    param v = 1          /* base swarm speed */
    param lambda = 0.1   /* density regulation */
    param eps = 1e-100   /* minimum gradient norm */
    param k_W = 0.1      /* degree of random motion */
    param k_P = 30       /* path deposition rate */
```

The parameter v defines the swarm speed, lambda (λ) controls the tradeoff between following the gradient and controlling the swarm density, eps (ϵ) is a small number to avoid division by zero, and k_W (k_W) controls randomness, explained below.

The next three equations define the vector field that directs the swarm's movement:

```
let U = del A
let S = ||U||
let V = [(v*U)/(S+eps)] - lambda*del[(C-1)^2] ...
  + [k_W DW^2]
```

The first line gives a name to the morphogen gradient ($\mathbf{U} = \nabla A$). Since the gradient may vary greatly in magnitude, we normalize it, and to this end, the scalar field S is defined $S = \|\mathbf{U}\|$. The final equation, which uses "..." for line continuation, defines the directive vector field:

$$\mathbf{V} = v\mathbf{U}/(S+\epsilon) - \lambda\nabla(C-1)^2 + k_W \text{Đ}W^2. \tag{3}$$

The attractant *versor*, or normalized vector, is $\mathbf{U}/S$, but we must avoid division where $S = 0$, so we use $\mathbf{U}/(S+\epsilon)$. The first term, then, of the definition of the velocity vector field $\mathbf{V}$ is the base speed times the gradient versor, that is, $v\mathbf{U}/(S+\epsilon)$, or "[(v*U)/(S+eps)]" in the programming language. The second term, $-\lambda\nabla(C-1)^2$, written "−lambda*del[(C−1)^2]," controls the density of the swarm to keep it compact but not too dense. This term (controlled by λ) directs motion in a direction that minimizes $|C-1|$ and therefore strives to keep the density near 1. The last term introduces some randomness into the swarm's movement to be more physically realistic and to help break symmetry. The programming notation "[k_W DW^2]" represents $k_W \text{Đ}W^2$, a two-dimensional normally-distributed random vector [7, 8, 9, 12].

The next equation in the behavior of the swarm substance describes the change in swarm concentration as a physical result of its velocity (which is controlled by the swarm agents):

```
D C = [t>5] -div[C*V]
```

The change in swarm concentration C is given by the negative divergence of the agent flux, $-\operatorname{div}(C\mathbf{V}) = -\nabla \cdot C\mathbf{V}$. The conditional factor [t>5] has the value 1 when time $t > 5$ and 0 otherwise; mathematically it is a Heaviside step function. This has the effect of suppressing movement for the first five time units in order to let the morphogen gradient stabilize before the swarm begins to move.

The final (partial) equation extends the definition of $\text{Đ}P$ to describe the deposition of path material:

```
D P += [t>5] k_P*[C*(1-P)]      /*path deposition*/
```

The source term "k_P * [C *(1−P)]," that is, $k_P C(1-P)$, describes how the swarm ($C > 0$) lays down path material at a rate k_P up to saturation ($P = 1$).

The [t>5] factor suppresses deposition for the first five time units so that the morphogen gradient can stabilize.

After the substance definitions comes the initialization of the various bodies involved in the simulation; they define the *initial preparation* of the morphogenetic process. The simplest body definition is the Goal (lines 62–64), which defines the small region of goal material from which the attractant diffuses and which the swarm will seek:

```
body Goal of goal_material:
  for −0.05 < x < 0.05, 0.9 < y < 0.95: G = 1
end
```

The Goal is on the x axis at $y = 0.925$ near the far limit of the space. Outside the initialized body, $G = 0$, because regions of any field that are not initialized are by default zero.

The swarm Cohort (lines 73–75) is initially at the origin of the path, which is on the x axis near the opposite side of the space from the Goal ($y = -0.925$):

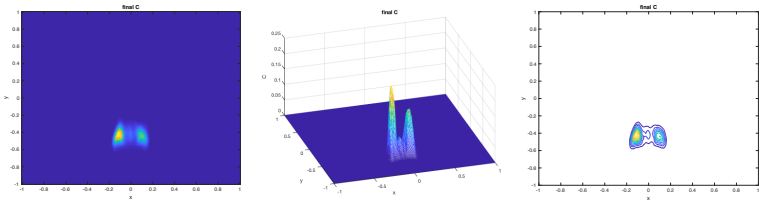
```
body Cohort of swarm:
  for −0.05 < x < 0.05, −0.95 < y < −0.9: C = 1
end
```

The path material P is laid down, of course, by the moving swarm, but we want it to avoid any paths that already exist. Therefore, for test purposes we define several pre-existing concentrations of path material to represent them (lines 66–71); since this is a 2D simulation, they are simply circular regions where $P = 1$:

```
body Obstacles of path_material:
  for (x, y) within 0.06 of (−0.1, 0.225): P = 1
  for (x, y) within 0.06 of (0.1, −0.225): P = 1
  for (x, y) within 0.06 of (0, −0.5): P = 1
  for (x, y) within 0.06 of (0, 0.5): P = 1
end
```

The final block in the program (lines 77–83) defines the visualization options:

```
visualization:
  display interval = 0.05
  display final P as colors limits (0, 0.5)
```



(a) colors (heatmap) (b) mesh (c) contours

FIGURE 1: Example displays of swarm density (at time $t = 5.5$). The swarm is splitting into two sub-swarms to go around an obstacle.

```
  display running C as colors limits (0, 0.5)
  display final P as mesh
  report Courant number for V
end
```

Fields can be displayed either at the end of the simulation, indicated by the keyword **final**, or while the simulation is executing, indicated by **running**. The fields are displayed at every time step unless a different **display interval** is defined, as in this example. In this case, the running display of C allows us to watch the movement of the swarm around obstacles toward the goal. The **colors** option displays a scalar field as a heat map (e.g., Figure 1a); **mesh** displays a scalar field as a 3D surface (e.g., Figure 1b), and **contours** displays a scalar field as a contour map (e.g., Figure 1c). The **limits** option clips values between the specified limits to ensure a consistent representation, especially for **running** displays. Vector fields can be displayed **as quivers** (little arrows, e.g., Figure 2). The **report** command, described later, allows useful numerical condition numbers to be monitored. A running display can be made into an mp4 movie with a command such as this:

```
make movie PathGrow of P as colors limits (0, 0.5)
```

This creates a movie file called "PathGrow.mp4" from a running display of P displayed as colors.

3 CONTINUOUS FLOCKING PATH FORMATION EXPERIMENTS

In this section we present the results of a series of experiments with the 2D continuous flocking approach to path creation, both to explore variations of

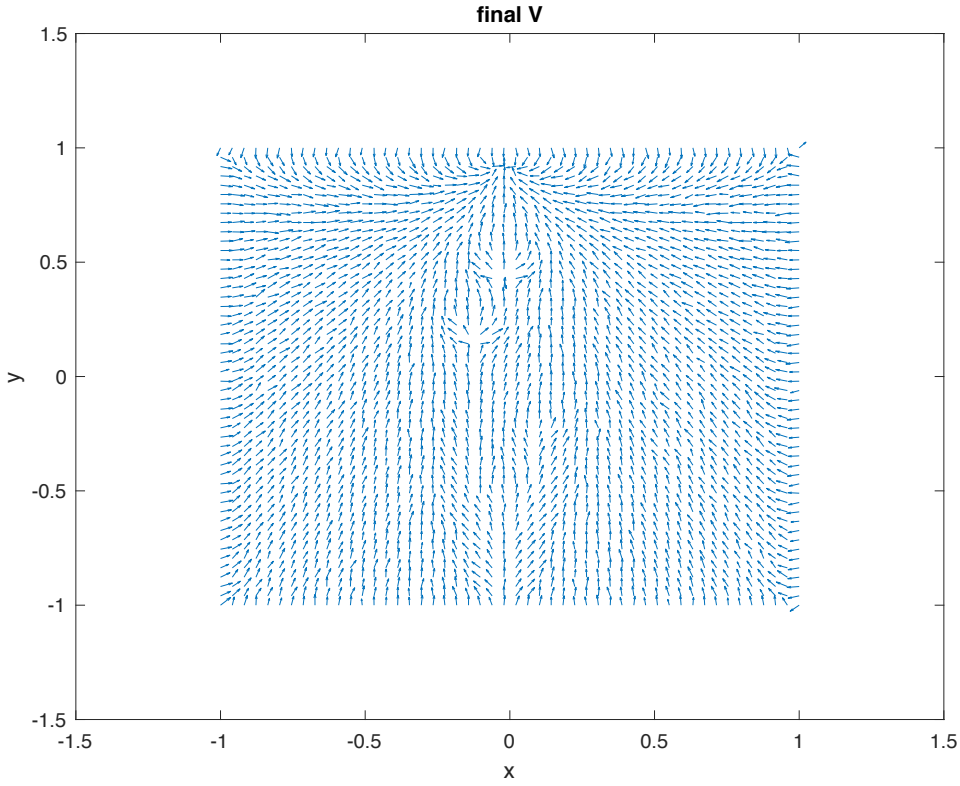
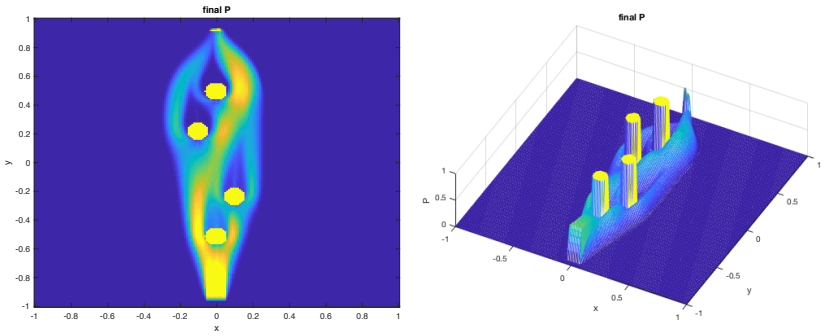


FIGURE 2: Vector field $\mathbf{V}$ ($\lambda = 0, t = 6$) displayed as quivers (50×50 grid).

the algorithm and to further illustrate the morphgen2D language. It is apparent that the algorithm involves many parameters, and so it is necessary to explore their effect on the outcome in order to adjust them to achieve particular purposes. Moreover, as in any algorithm, we may entertain different ways of accomplishing various purposes, which might work better or worse. The basis for these experiments is the algorithm presented in Appendix A and described in the preceding section. The path material deposited in a typical run is shown in Figure 3. Nominal parameters for the simulations are summarized in Table 1.

Figure 4 shows the effect of λ, which controls the relative importance of maintaining a density $C \approx 1$, on the structure of the paths. Figure 4a shows the case $\lambda = 0$, that is, there is no constraint on the density, and the swarm



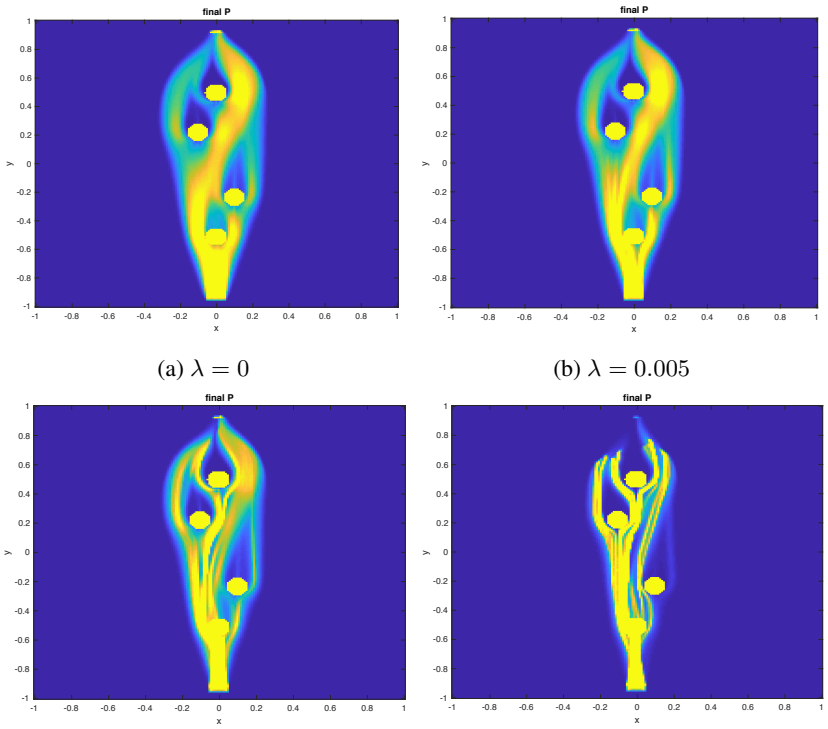
(a) colors plot

(b) mesh plot

FIGURE 3: Final concentration of path material.

Param.	Value	Meaning
T	6.75	duration
Δs	0.01	spatial resolution
Δt	0.001	temporal resolution
d_A	0.03	attractant diffusion constant
τ_A	100	attractant decay time constant
τ_P	0.2	attractant absorption time constant
k_G	100	attractant release rate from goal substance
v	1	base swarm speed
λ	0.03	importance of swarm density
ϵ	10^{-100}	minimum attractant gradient magnitude
k_W	0.3	amount of random motion
k_P	30	path deposition rate

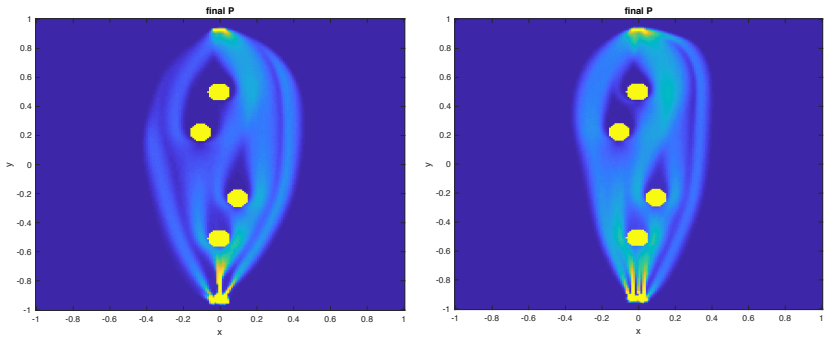
TABLE 1: Nominal Parameter Values



(a) $\lambda = 0$ (b) $\lambda = 0.005$

(c) $\lambda = 0.01$ (d) $\lambda = 0.02$

FIGURE 4: Effects of λ on path structure ($k_P = 30, t = 6.75$, color limits = $[0, 0.5]$).



(a) $\lambda = 10^{-8}$ (b) $\lambda = 10^{-10}$

FIGURE 5: Path densities resulting from normalizing sum of morphogen and density gradients (color limits = $[0, 0.5]$).

is moving entirely under the influence of the morphogen gradient. Figures 4b to 4d show the paths created with successively larger values of λ, and it is apparent that they create narrower and better defined paths.

The simulation becomes numerically unstable for $\lambda \geq 0.05$, probably because the density-driven gradient is causing the total velocity to become too great. The morphgen language includes visualization commands to report various numerical condition numbers. In this case we used the morphgen statement

```
report Courant number for V
```

to display at every display interval the Courant number

$$C_r = (\max_{x,y} |V_{\mathrm{x}}(x, y)| + |V_{\mathrm{y}}(x, y)|)\Delta t/\Delta s$$

(for time step size Δt and mesh spacing Δs). Stable simulations had Courant numbers $C_r \leq 0.54$, but for $\lambda = 0.05$ the simulation was unstable with $C_r > 0.59$. Halving the time step to $\Delta t = 0.0005$ resulted in a stable $\lambda = 0.05$ simulation with $C_r \leq 0.33$.

Since the density gradient is added to the normalized morphogen gradient (Eq. 3), a large density gradient can result in a high velocity, causing numerical instability and possibly physically impossible behavior. We can compare the magnitude of the morphogen gradient, constrained to the speed v, with

the magnitude of the density-driven gradient, $\|\nabla(C-1)^2\|$:

$$\begin{aligned}\|\nabla(C-1)^2\|^2 &= \left[\frac{\partial(C-1)^2}{\partial x}\right]^2 + \left[\frac{\partial(C-1)^2}{\partial y}\right]^2 \\ &= \left(2\frac{\partial C}{\partial x}\right)^2 + \left(2\frac{\partial C}{\partial y}\right)^2 = 4\nabla^2 C.\end{aligned}$$

Hence, the relative magnitudes of the morphogen and density components of the velocity are v and $2\lambda\sqrt{\nabla^2 C}$ respectively, and therefore large density gradients might lead to excessive velocities. An alternative is to add the density gradient to the morphogen gradient before normalization, so that the resulting total velocity is limited. This is accomplished by changing the definition of the $\mathbf{U}$ and $\mathbf{V}$ vector fields as follows:

```
let U = del A - lambda*del[(C-1)^2]
let S = ||U||
let V = [(v*U)/(S+eps)] + [k_W DW^2]
```

Figure 5 shows two typical runs for λ values that showed some evidence of density control. It can be seen that initially the swarm divided into several well-defined streams, but that these soon spread out and became diffuse. The explanation seems to be that the morphogen gradient at a distance r from the goal is

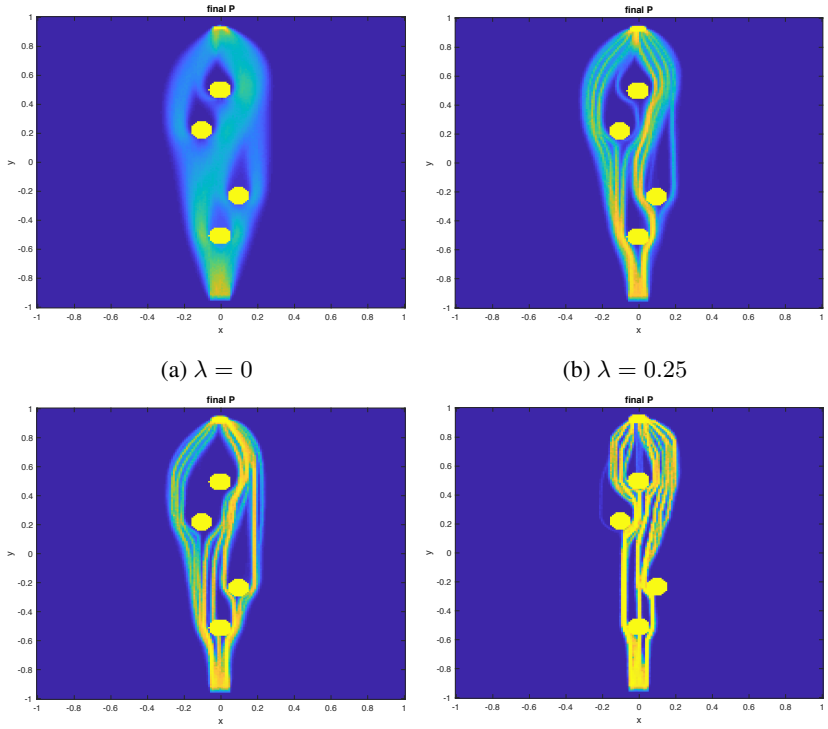
$$\frac{\mathrm{d}A(r)}{\mathrm{d}r} = \frac{\mathrm{d}}{\mathrm{d}r} k_G \exp\left(-\frac{r}{\sqrt{d_A \tau_A}}\right) = -\frac{k_G}{\sqrt{d_A \tau_A}} \exp\left(-\frac{r}{\sqrt{d_A \tau_A}}\right).$$

Therefore the gradient varies with distance from the goal, and so the relative contributions of the morphogen and density gradients to the velocity will vary with location. This seems to be why the density limit is effective near the origin, that is, far from the goal, and becomes ineffective as the goal is approached. In conclusion, including the density gradient before normalization does not appear to be a useful strategy.

The preceding problem can be avoided by normalizing the morphogen and density gradients separately before combining them; in this way there is a consistent balance between the two gradients throughout the space. We let $\mathbf{W} = \nabla(C-1)^2$ be the density gradient and $T = \|\mathbf{W}\|$ be its magnitude. Then,

$$\mathbf{V} = v[(1-\lambda)\mathbf{U}/(S+\epsilon) - \lambda\mathbf{W}/(T+\epsilon) + k_W \text{Đ}W^2] \tag{4}$$

averages the versors of the morphogen and density gradients, with the weight controlled by λ. This is accomplished by the following morphgen code:



(a) $\lambda = 0$ (b) $\lambda = 0.25$

(c) $\lambda = 0.35$ (d) $\lambda = 0.5$

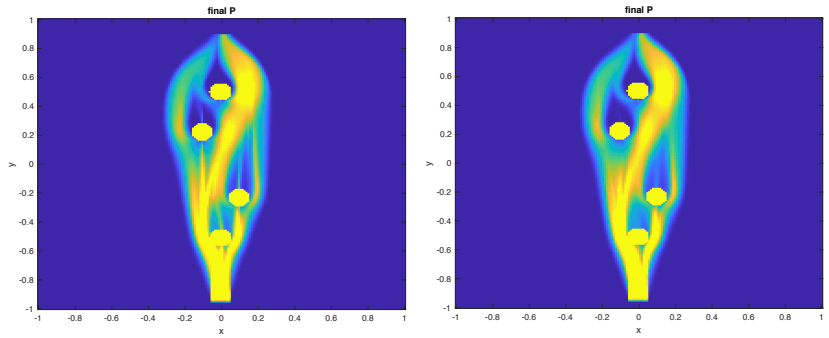
FIGURE 6: Path densities resulting from normalizing morphogen and density gradients before combination ($T = 14$, color limits =$[0, 1]$).

```
let U = del A
let S = ||U||
let W = del[(C-1)^2]
let T = ||W||
let V = v * ((1 - lambda) * [U/(S+eps)] ...
            - lambda * [W/(T+eps)] + [k_W DW^2])
```

With this change, $\lambda > 0$ values do control the density, causing the swarm to break up into small compact groups and lay down paths of relatively constant width (Figure 6). This also delays arrival at the destination, since the effective speed of following the morphogen is $v(1 - \lambda)$, so the simulations were run longer, $T = 14$ time units.

Equation 3 for $\mathbf{V}$ (p. 7) includes a random element $k_W \mathrm{D}W^2$, the purpose of which is to be physically realistic (motion cannot be controlled perfectly) and to break a symmetry that otherwise leads to unrealistic results. The problem with symmetry is that on the "downwind" side of obstacles there are regions where the gradients resulting from morphogen diffusing around the left and right sides balance each other, so the resulting velocity vector is aimed directly at the obstacle. (This can be seen clearly in the velocity vector field, Figure 2.) Therefore, instead of going around the obstacles, a small part of the swarm "tunnels" through it (since the simulation does not model the fact that the obstacles are solid and therefore impenetrable). This can be seen in Figure 7a, in which $k_W = 0$ and therefore there is no randomness: narrow streams drive directly into the obstacles and emerge on the other sides. Progressively larger amounts of randomness ($k_W = 0.1, 0.2, 0.3$) decrease and ultimately eliminate the tunneling (Figures 7b–7d). The value $k_W = 0.3$ is used in subsequent experiments.

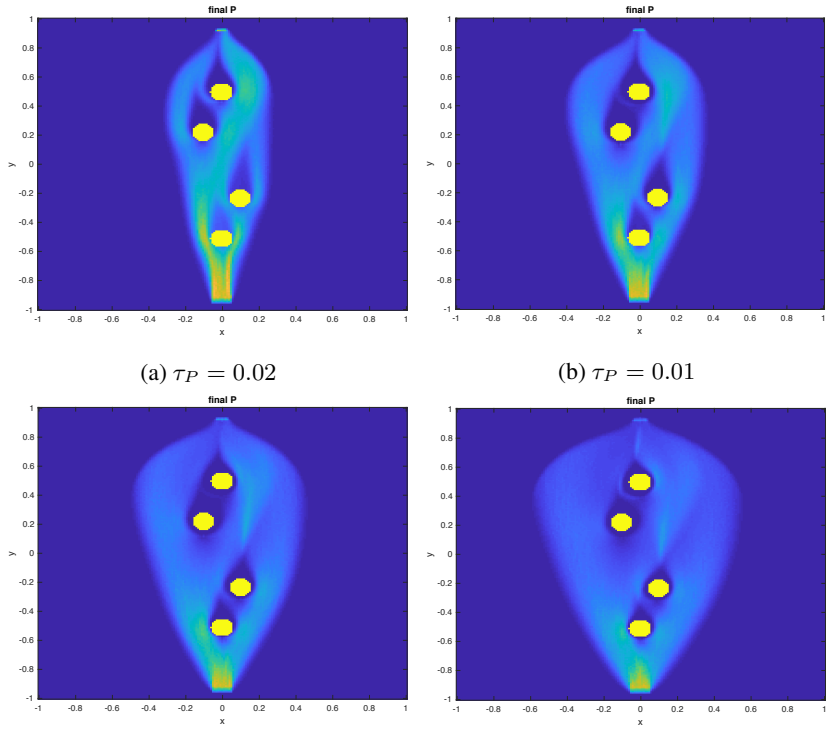
In many of the simulations, the swarms pass very close to the obstacles, and so we have conducted several experiments to control the *margins* around the obstacles. The principal mechanism for obstacle avoidance is the absorption or degradation of attractant morphogen by path material, represented by the $-PA/\tau_P$ term in Eq. 2. Smaller values of the time constant τ_P lead to quicker elimination of attractant, which does indeed lead to larger margins. Figure 8 shows path densities resulting from several different values of τ_P, and it is apparent that smaller time constants lead to larger margins. In addition to larger margins, it is also apparent that small time constants lead to a larger spread in the path material, resulting from an increasing spread in the swarm. We conjecture that this is because the obstacles are absorbing attrac-



(a) $k_W = 0$ (b) $k_W = 0.1$

(c) $k_W = 0.2$ (d) $k_W = 0.3$

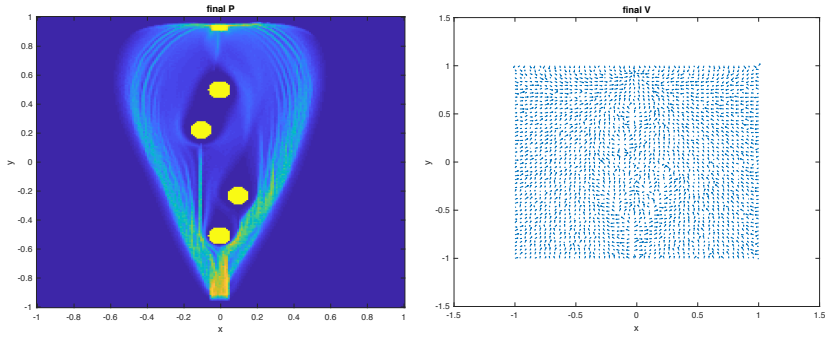
FIGURE 7: Effect of randomness on path formation. Sufficient randomness eliminates "tunneling" through obstacles.



(a) $\tau_P = 0.02$ (b) $\tau_P = 0.01$

(c) $\tau_P = 0.005$ (d) $\tau_P = 0.0025$

FIGURE 8: Margins around obstacles resulting from various time constants τ_P for attractant elimination ($\lambda = 0.02$).

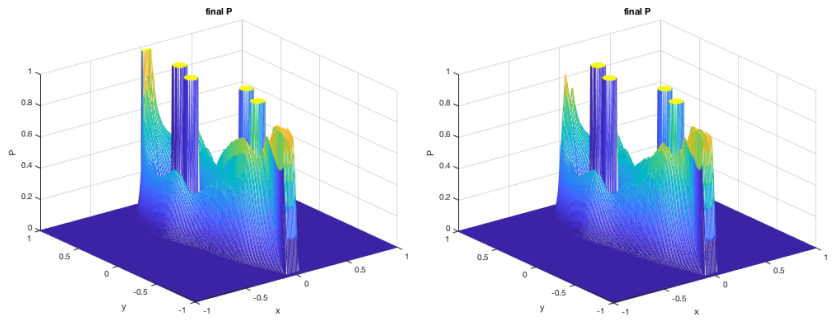


(a) P path density field (b) $\mathbf{V}$ velocity vector field

FIGURE 9: Path density and velocity field with rapid attractant elimination ($\tau_P = 0.025, \lambda = 0.02$).

tant from all directions equally, and that this generally steers the swarm away from them. This can be seen in Figure 9, which shows the path density and the velocity field that produced it.

In the morphogenetic process as programmed, the swarm will continue to flow into the goal region, limited only by the back pressure caused by a density $C > 1$. At this point the velocity becomes unstable, since the morphogen gradient is effectively zero and the swarm clusters in and around the goal region, laying down more path material all the time, until it reaches saturation, as can be seen in Figure 10a. (Recall that the equation for $Ð P$ causes it to saturate at $P = 1$.) This accumulation in the goal region may be undesirable, and one solution is to have the goal material G rapidly absorb the swarm C, which we can accomplish by adding a partial equation $Ð C \mathrel{-}= k_C G C$ to the goal substance (since it represents action by the goal agents):



(a) $k_C = 0$ (b) $k_C = 100$

FIGURE 10: Effect of swarm absorption by goal material ($t = 10$). (a) No absorption: the path material saturates at $P = 1$ in the goal region. (b) Absorption limits path density in the goal region to $P \approx 0.8$.

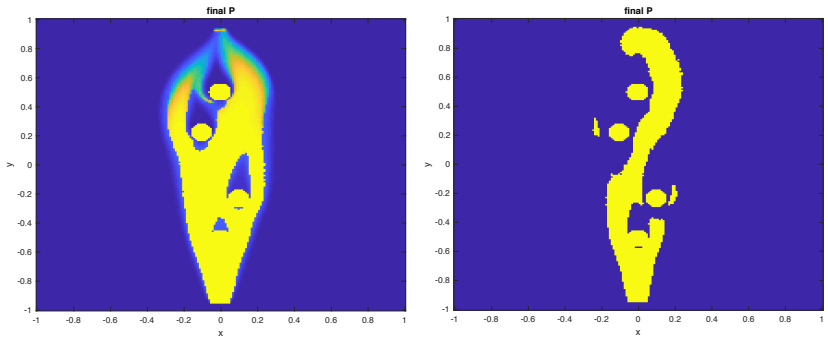
```
param k_C = 100        /* swarm absorption rate */
D C -= k_C*[G*C]       /* absorb swarm at goal */
```

Notice that in Figure 10a the path density at the goal has saturated at its maximum value $P = 1$, whereas in Figure 10b with absorption ($k_C = 100$) it reached only $P \approx 0.8$.

It is apparent that the paths laid down are not of uniform density (e.g., Figures 3, 4, 10). Since the paths represent bundles of fibers, some variation in density across the width of a path is unproblematic, but we expect consistent density along the trajectory from the origin to the destination. One solution is to make the path material autocatalytic; that is, the presence of path material catalyzes the creation of new path material at a rate a_P up to a maximum. To avoid very low densities of path material triggering autocatalysis, which fills a lot of the space with path material, autocatalysis is triggered by path material only above a specified threshold θ_P. Autocatalysis is accomplished by adding the partial equation $\text{Đ}P \mathrel{+}= [P > \theta_P]P(1-P)$ to the behavior of the path substance. By itself, this leaves the low density path material in the environment (Figure 11a), but it can be eliminated by a decay term $-P/t_P$ operative for below-threshold densities. The autocatalysis and decay terms are combined in the partial equation:

$$\text{Đ}P \mathrel{+}= a_P[P > \theta_P]P(1-P) - [P \leq \theta_P]P/t_P. \tag{5}$$

The program code to accomplish this is:



(a) $\theta_P = 0.2$, no decay (b) $\theta_P = 0.3$, with decay ($t_P = 1$)

FIGURE 11: Use of autocatalysis and decay to control path density.

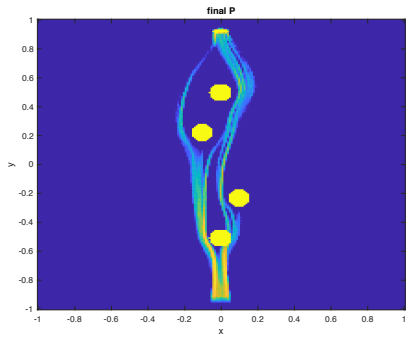
```
param theta_P = 0.3  /* autocatalytic threshold */
param a_P = 20       /* autocatalytic rate */
param t_P = 1        /* path decay time const. */
// autocatalysis:
D P += [t>5](a_P*[P>theta_P] [P*(1-P)] ...
            - [P<=theta_P] P/t_P)
```

Figure 11b shows an example with both autocatalysis and decay; the simulation was run for 10 time units to allow the processes to complete. The path is quite wide, there are a few isolated islands of path material, and there seems to be no gap between the path and the first obstacle.

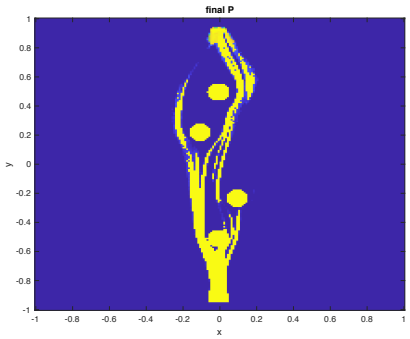
An alternative approach to controlling path density is to have the swarm do *quorum sensing* and only lay down path material if the swarm density is above a threshold; in this way, low density areas of the swarm will not produce path material. Adding a swarm threshold $[C > \theta_C]$ governing path deposition to the $\eth P$ equation produces well-defined paths, but the density is variable (Figure 12a). This can be avoided by combining quorum sensing for path deposition with autocatalysis and decay of the path material (Eq. 5) to obtain:

$$\begin{aligned} \eth P &\mathrel{+}= a_P[P > \theta_P]P(1-P) - [P \leq \theta_P]P/t_P, \\ \eth P &\mathrel{+}= [t > 5]\; k_P[C > \theta_C]C(1-P). \end{aligned}$$

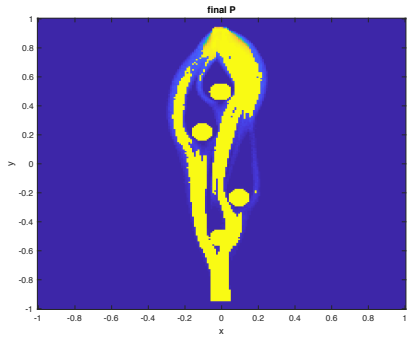
The first partial equation becomes part of the path substance, since it is part of its behavior; the second partial equation becomes part of the swarm behavior,



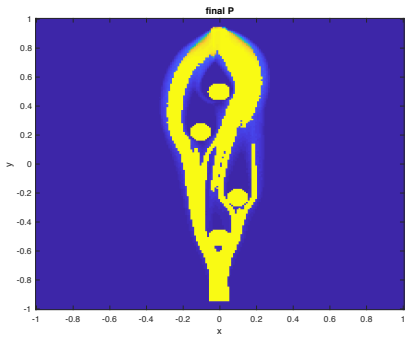
(a) $\theta_C = 0.25$, without autocatalysis or decay



(b) $\theta_C = 0.25, a_P = 20, \tau_P = 0.25$



(c) $\theta_C = 0.05, a_P = 20, \tau_P = 0.25$



(d) $\theta_C = 0.02, a_P = 20, \tau_P = 0.25$

FIGURE 12: Path formation with swarm quorum sensing.

Param.	Value	Meaning
T	10	duration
Δs	0.01	spatial resolution
Δt	0.001	temporal resolution
d_A	0.03	attractant diffusion constant
τ_A	100	attractant decay time constant
τ_P	0.1	attractant absorption time constant
k_G	100	attractant release rate from goal substance
v	1	base swarm speed
λ	0.5	importance of swarm density
ϵ	10^{-100}	minimum attractant gradient magnitude
k_W	0.7	amount of random motion
k_P	30	path deposition rate
a_P	20	path autocatalysis rate
θ_P	0.3	path autocatalysis threshold
t_P	1	path decay time constant
k_C	100	swarm absorption rate

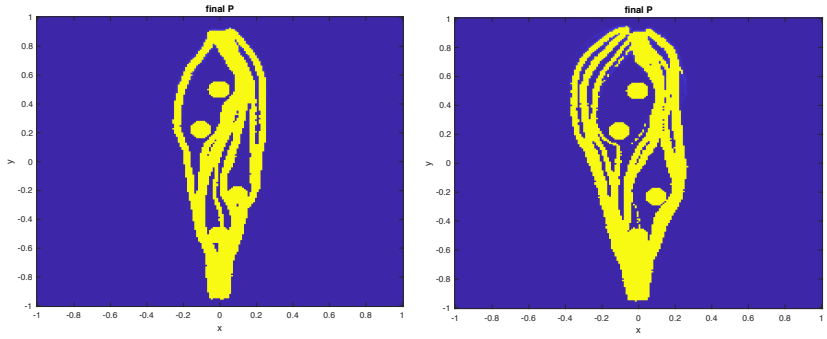
TABLE 2: Revised Parameter Values

since the agents control path deposition by quorum sensing $[C > \theta_C]$. In above-quorum regions, the path material will increase to saturation through autocatalysis (Figs. 12b–12d). Lower quorum thresholds θ_C produce thicker paths. These simulations were run for a duration $T = 10$ to allow the processes to complete. In general, quorum sensing with autocatalysis seems to produce discontinuous and irregular paths.

4 REVISED NOMINAL PARAMETERS

Drawing on the preceding experiments, we collect in Table 2 the parameters that give good results. By default, we use prenormalization of the morphogen and density-control gradients (Eq. 4, p. 14), and to promote uniform path density, we use autocatalysis with decay (Eqs. 5, p. 20), but not quorum sensing. Figure 13 shows two simulations: Figure 13a has $\tau_P = 0.2$, which has tunneling through the first obstacle, and Figure 13b has a quicker $\tau_P = 0.1$, which eliminates the tunneling.

To see how well these parameters generalize, we ran simulations with additional obstacles and different origin and destination (Figure 14). These simulations were also run at higher resolution: $\Delta s = 0.005, \Delta t = 0.0005$. Figure



(a) $\tau_P = 0.2$ (b) $\tau_P = 0.1$

FIGURE 13: Simulations based on revised parameter values (Table 2). The smaller τ_P eliminates tunneling.

14a uses the parameters in Table 2; the paths are largely continuous, but the high $\lambda = 0.5$ has caused the streams to separate a little. Therefore, Figure 14b shows the result with a smaller $\lambda = 0.4$; it has fewer gaps, but the leftmost path is quite thin. A further decrease to $\lambda = 0.3$ does lead to more complete paths, except for the path on the left, which is broken (Figure 14c). Figure 14d shows that this can be filled in by lowering the autocatalysis threshold θ_P from 0.3 to 0.25 (which is perhaps a better default value).

5 THREE-DIMENSIONAL SIMULATION

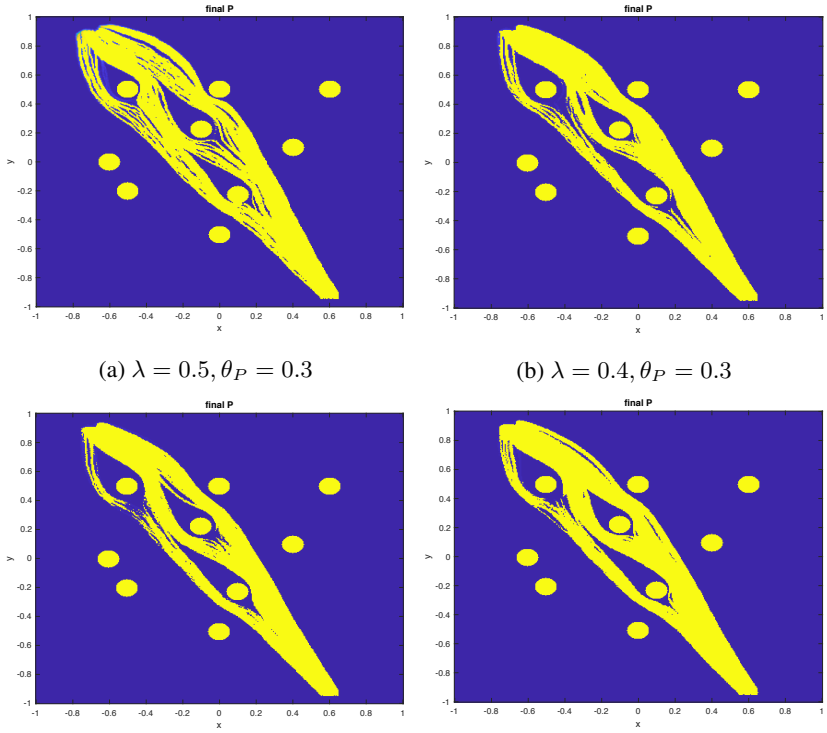
Finally, we illustrate a three-dimensional artificial morphogenesis simulation. The 3D program is essentially the same as the 2D version (Sec. 2); it is written in morphgen3D, which is nearly identical to morphgen2D. Figure 15 shows the path created by a simulation with the parameters shown and using this version of the velocity equation:

```
let V = [(v*U)/(S+eps)] - lambda*del[(C-1)^2]
```

The near identity of the 2D and 3D programs allows experience with the 2D simulations to be transferred to the more realistic 3D simulations.

6 CONCLUSIONS

We have illustrated *morphgen2D*, an artificial morphogenesis programming language, by explaining a continuous flocking algorithm to control extremely



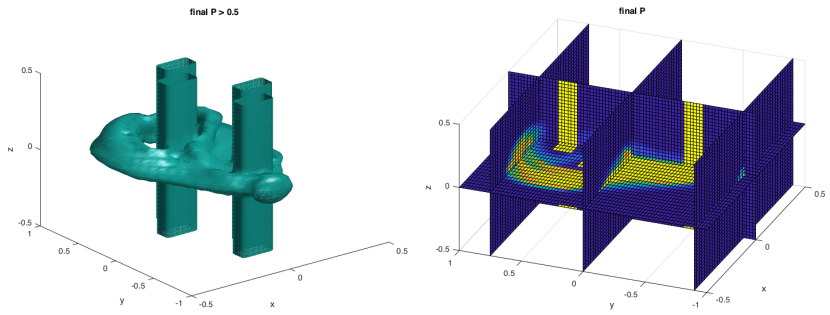
(a) $\lambda = 0.5, \theta_P = 0.3$

(b) $\lambda = 0.4, \theta_P = 0.3$

(c) $\lambda = 0.3, \theta_P = 0.3$

(d) $\lambda = 0.3, \theta_P = 0.25$

FIGURE 14: Simulations at higher resolution ($\Delta s = 0.005, \Delta t = 0.0005$) with different obstacles, origin (lower right), and destination (upper left).



(a) final $P > 0.5$ (b) final P density

FIGURE 15: 3D simulation of path formation ($d_A = 0.03, \tau_A = 100, k_G = 100, \tau_P = 0.2, v = 1, \lambda = 0.03, \epsilon = 10^{-100}, \tau_C = 0.01, k_P = 30, T = 6.5, \Delta s = 0.01, \Delta t = 0.001$).

large swarms of agents creating paths (such as artificial nerve fibers) between designated locations. We then described a series of experiments exploring the parameter space and algorithm variants in order to produce acceptable paths. We found that attractant and density-control gradients should be normalized before combination, the relative weights of these gradients can be used to control path definition, autocatalysis and decay can be used to ensure uniform path density, swarm absorption by the goal region avoids saturation, and moderate randomness can break undesirable symmetries. Quorum sensing, rapid attractant absorption by path material to increase margins, and alternative gradient normalization approaches were less successful. Finally, we showed that the 2D simulation could be converted to a 3D simulation with minimal changes. More generally, we showed how an artificial morphogenesis process, expressed as partial differential equations, could be expressed in a formal morphogenetic programming language and compiled into simulation software that could be used to refine the process.

7 ACKNOWLEDGEMENTS

I am grateful to Allen McBride for suggesting normalization of the morphogen and density gradients separately before combining them (Eq. 4). This article is a revision of a previous technical report [17].

A 2D CONTINUOUS FLOCKING PROGRAM

```
#include "morphgen2D.smac"
\alpha "_." // allow in variable names and numbers

// Continuous Flocking Path Generation

morphogenetic program cont_flock:

simulation parameters:
  space: -1 < x < 1, -1 < y < 1
  duration = 6.75
  spatial resolution = 0.01
  temporal resolution = 0.001
end

substance morphogen:
    scalar field A
  behavior:
    param d_A = 0.03    /* attractant diffusion constant */
    param tau_A = 100   /* attractant decay time constant */
    D A += d_A * del^2 A - A/tau_A
end

substance path_material:
    scalar field P
  behavior:
    param tau_P = 0.2   /* attractant absorption time constant */
    D P += 0            /* passive path material */
    D A -= [P*A]/tau_P /* path absorbs attractant */
end

substance swarm:
    scalar fields:
      C  /* swarm concentration */
      S  /* magnitude of morphogen gradient */
    end
    vector fields:
      U  /* morphogen gradient */
```

```
38        V  /* swarm velocity */
39      end
40    behavior:
41      param v = 1             /* base swarm speed */
42      param lambda = 0.03     /* density regulation */
43      param eps = 1e-100      /* minimum gradient norm */
44      param k_W = 0.1         /* degree of random motion */
45      param k_P = 30          /* path deposition rate */
46      let U = del A
47      let S = ||U||
48      let V = [(v*U)/(S+eps)] - lambda*del[(C-1)^2] ...
49        + [k_W DW^2]
50      D C = [t>5] -div[C*V]
51      D P += [t>5] k_P*[C*(1-P)] /* path deposition */
52  end
53
54  substance goal_material:
55      scalar field G
56    behavior:
57      param k_G = 100         /* attractant release rate */
58      D G = 0                 /* G field is fixed */
59      D A += k_G*[G*(1-A)] /* goal emits attractant */
60  end
61
62  body Goal of goal_material:
63    for -0.05 < x < 0.05, 0.9 < y < 0.95: G = 1
64  end
65
66  body Obstacles of path_material:
67    for (x, y) within 0.06 of (-0.1, 0.225): P = 1
68    for (x, y) within 0.06 of (0.1, -0.225): P = 1
69    for (x, y) within 0.06 of (0, -0.5): P = 1
70    for (x, y) within 0.06 of (0, 0.5): P = 1
71  end
72
73  body Cohort of swarm:
74    for -0.05 < x < 0.05, -0.95 < y < -0.9: C = 1
75  end
76
```

```
77 visualization:
78   display interval = 0.05
79   display final P as colors limits(0, 0.5)
80   display running C as colors limits(0, 0.5)
81   display final P as mesh
82   report Courant number for V
83 end
84
85 end program
```

B MORPHGEN2D SYNTAX

Notation: Square brackets surround optional items; curly braces group items. (When brackets are terminal symbols, they are in boldface.) Superscript * means zero or more repetitions, superscript + means one or more repetitions. ⟨comment⟩s can appear anywhere whitespace is allowed (generally, between tokens). Line continuation of an expression is indicated by " ... ", which is treated as whitespace.

⟨program⟩ ::= **morphogenetic program** ⟨name⟩ :
⟨sim params⟩
⟨substance⟩*
⟨body⟩*
⟨visualization⟩
end program

⟨sim params⟩ ::= **simulation parameters** : ⟨param⟩* **end**

⟨param⟩ ::= **duration** = ⟨num⟩⟨newline⟩
| **temporal resolution** = ⟨num⟩⟨newline⟩
| **space** ⟨num⟩ < **x** < ⟨num⟩, ⟨num⟩ < **y** < ⟨num⟩⟨newline⟩
| **spatial resolution** = ⟨num⟩⟨newline⟩
| **save** ⟨name⟩+ **to** ⟨filename⟩⟨newline⟩
| **load** ⟨name⟩+ **from** ⟨filename⟩⟨newline⟩
| ⟨log params⟩

⟨substance⟩ ::= **substance** ⟨name⟩ :
⟨variable⟩*
behavior :

$\langle$equation$\rangle^*$

end

$\langle$variable$\rangle$ ::= **scalar field** $\langle$name$\rangle\langle$newline$\rangle$

| **vector field** $\langle$name$\rangle\langle$newline$\rangle$

| **scalar fields** : $\langle$newline$\rangle\langle$name list$\rangle$ **end**

| **vector fields** : $\langle$newline$\rangle\langle$name list$\rangle$ **end**

$\langle$name list$\rangle$ ::= $\{\langle$name$\rangle\langle$newline$\rangle\}^+$

$\langle$equation$\rangle$ ::= **D** $\langle$name$\rangle$ $[+|-] = \langle$expr$\rangle\langle$newline$\rangle$

| **let** $\langle$name$\rangle = \langle$expr$\rangle\langle$newline$\rangle$

| **param** $\langle$name$\rangle = \langle$expr$\rangle\langle$newline$\rangle$

| $\langle$log params$\rangle$

$\langle$expr$\rangle$ ::= $\langle$primitive$\rangle[\langle$operator$\rangle\langle$primitive$\rangle]^*$

$\langle$operator$\rangle$ ::= $+ \mid - \mid * \mid / \mid \hat{}$

$\langle$primitive$\rangle$ ::= $\langle$name$\rangle \mid \langle$num$\rangle \mid (\langle$expr$\rangle) \mid \langle$special$\rangle$

$\langle$special$\rangle$ ::= **del** $\langle$primitive$\rangle$

| **del^2** $\langle$primitive$\rangle$

| **div** $\langle$primitive$\rangle$

| $||\langle$expr$\rangle||$

| **[**$\langle$primitive$\rangle > \langle$primitive$\rangle$**]** $\langle$primitive$\rangle$

| **[**$\langle$primitive$\rangle >= \langle$primitive$\rangle$**]** $\langle$primitive$\rangle$

| **[**$\langle$primitive$\rangle < \langle$primitive$\rangle$**]** $\langle$primitive$\rangle$

| **[**$\langle$primitive$\rangle <= \langle$primitive$\rangle$**]** $\langle$primitive$\rangle$

| **[**$\langle$primitive$\rangle * \langle$primitive$\rangle$**]**

| **[**$\langle$primitive$\rangle$ / $\langle$primitive$\rangle$**]**

| **[**$\langle$primitive$\rangle \hat{} \langle$primitive$\rangle$**]**

| **[**$\langle$primitive$\rangle$ **DW** $\hat{} \langle$primitive$\rangle$**]**

$\langle$body$\rangle$ ::= **body** $\langle$name$\rangle$ **of** $\langle$name$\rangle$: $\langle$definition$\rangle^*$ **end**

$\langle$definition$\rangle$ ::= **for** $\langle$region$\rangle$: $\langle$name$\rangle = \langle$expr$\rangle\langle$newline$\rangle$

| **for** $\langle$region$\rangle$: $\langle$newline$\rangle\langle$init$\rangle^*$ **end**

$\langle$region$\rangle$ ::= $\langle$expr$\rangle < \langle$name$\rangle < \langle$expr$\rangle, \langle$expr$\rangle < \langle$name$\rangle < \langle$expr$\rangle$

| $(\langle$name$\rangle, \langle$name$\rangle)$ **within** $\langle$expr$\rangle$ **of** $(\langle$expr$\rangle, \langle$expr$\rangle)$

$$
\begin{array}{rcl}
\langle \text{init} \rangle & ::= & \langle \text{name} \rangle = \langle \text{expr} \rangle \langle \text{newline} \rangle \\
\langle \text{visualization} \rangle & ::= & \textbf{visualization } \langle \text{vis command} \rangle^+ \textbf{ end} \\
\langle \text{vis command} \rangle & ::= & \textbf{display } \langle \text{time} \rangle \ \langle \text{primitive} \rangle \textbf{ as } \langle \text{kind} \rangle \\
& | & \textbf{display running code } \langle \text{target code} \rangle \textbf{ end code} \\
& | & \textbf{make movie } \langle \text{filename} \rangle \textbf{ of } \langle \text{primitive} \rangle \textbf{ as } \langle \text{kind} \rangle \langle \text{newline} \rangle \\
& | & \textbf{record parameters in } \langle \text{filename} \rangle \langle \text{newline} \rangle \\
& | & \langle \text{stability report} \rangle \\
\langle \text{time} \rangle & ::= & \textbf{running} \mid \textbf{final} \\
\langle \text{kind} \rangle & ::= & \{\textbf{mesh} \mid \textbf{contours} \mid \textbf{colors}\} \ [\textbf{limits } (\langle \text{expr} \rangle, \langle \text{expr} \rangle)] \\
& | & \textbf{quivers } [\textbf{grid } (\langle \text{expr} \rangle, \langle \text{expr} \rangle)] \\
\langle \text{stability report} \rangle & ::= & \textbf{report diffusion number for } \langle \text{primitive} \rangle \\
& | & \textbf{report Courant number for } \langle \text{primitive} \rangle \\
& | & \textbf{report Peclet number for } \langle \text{primitive} \rangle \textbf{ and } \langle \text{primitive} \rangle \\
\langle \text{log params} \rangle & ::= & \textbf{log params } \langle \text{name} \rangle \ [, \ \langle \text{name} \rangle]^* \ \langle \text{newline} \rangle \\
\langle \text{name} \rangle & ::= & \langle \text{letter} \rangle \ [\ \langle \text{letter} \rangle \mid \langle \text{digit} \rangle \mid \_ \ ]^* \\
\langle \text{num} \rangle & ::= & [-] \langle \text{digit} \rangle^* [. \langle \text{digit} \rangle^*] \\
\langle \text{comment} \rangle & ::= & /* \ \langle \text{characters} \rangle \ */ \\
& | & // \ \langle \text{characters} \rangle \langle \text{newline} \rangle
\end{array}
$$

REFERENCES

[1] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. (May 2000). Amorphous computing. *Commun. ACM*, 43(5):74–82.

[2] D. A. Beysens, G. Forgacs, and J. A. Glazier. (2000). Cell sorting is analogous to phase ordering in fluids. *Proc. Nat. Acad. Sci. USA*, 97:9467–9471.

[3] J. A. Carrillo, M. Fornasier, G. Toscani, and F. Vecil. (2010). Particle, kinetic, and hydrodynamic models of swarming. In G. Naldi, L. Pareschi, and G. Toscani, editors, *Mathematical Modeling of Collective Behavior in Socio-Economic and Life Sciences*, pages 297–336. Birkhāuser, Boston.

[4] Y.-L. Chuang, M. R. D'Orsogna, D. Marthaler, A. L. Bertozzi, and L. S. Chayes. (2007). State transitions and the continuum limit for a 2d interacting, self-propelled particle system. *Physica D*, 232:33–47.

[5] Gabor Forgacs and Stuart A. Newman. (2005). *Biological Physics of the Developing Embryo*. Cambridge University Press, Cambridge, UK.

[6] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. (June 2005). Programmable matter. *Computer*, 38(6):99–101.

[7] Bruce J. MacLennan. (2009). Preliminary development of a formalism for embodied computation and morphogenesis. Technical Report UT-CS-09-644, Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN.

[8] Bruce J. MacLennan. (2010). Models and mechanisms for artificial morphogenesis. In F. Peper, H. Umeo, N. Matsui, and T. Isokawa, editors, *Natural Computing*, Springer series, Proceedings in Information and Communications Technology (PICT) 2, pages 23–33, Tokyo. Springer.

[9] Bruce J. MacLennan. (2010). Morphogenesis as a model for nano communication. *Nano Communication Networks.*, 1(3):199–208.

[10] Bruce J. MacLennan. (2010). The U-machine: A model of generalized computation. *International Journal of Unconventional Computing*, 6(3–4):265–283.

[11] Bruce J. MacLennan. (2011). Artificial morphogenesis as an example of embodied computation. *International Journal of Unconventional Computing.*, 7(1–2):3–23.

[12] Bruce J. MacLennan. (2012). Embodied computation: Applying the physics of computation to artificial morphogenesis. *Parallel Processing Letters*, 22(3):1240013.

[13] Bruce J. MacLennan. (June 2012). Molecular coordination of hierarchical self-assembly. *Nano Communication Networks*, 3(2):116–128.

[14] Bruce J. MacLennan. (2014). Coordinating massive robot swarms. *International Journal of Robotics Applications and Technologies*, 2(2):1–19.

[15] Bruce J. MacLennan. (2015). The morphogenetic path to programmable matter. *Proceedings of the IEEE*, 103(7):1226–1232.

[16] Bruce J. MacLennan. (2018). Coordinating swarms of microscopic agents to assemble complex structures. In Ying Tan, editor, *Swarm Intelligence, Vol. 1: Principles, Current Algorithms and Methods*, PBCE 119, chapter 20, pages 583–612. Institution of Engineering and Technology.

[17] Bruce J. MacLennan. (2018). Path creation by continuous flocking as an example of a morphogenetic programming language. Faculty Publications and Other Works — EECS. http://trace.tennessee.edu/utk_elecpubs/24, University of Tennessee Department of Electrical Engineering and Computer Science.

[18] Bruce J. MacLennan. (2018). The Synmac syntax macroprocessor: Introduction and manual, version 5. Faculty Publications and Other Works — EECS. http://trace.tennessee.edu/utk_elecpubs/23, University of Tennessee Department of Electrical Engineering and Computer Science.

[19] Hans Meinhardt. (1982). *Models of Biological Pattern Formation*. Academic Press, London.

[20] S. Murata and H. Kurokawa. (March 2007). Self-reconfigurable robots: Shape-changing cellular robots can exceed conventional robot flexibility. *IEEE Robotics & Automation Magazine*, pages 71–78.

[21] Radhika Nagpal, Attila Kondacs, and Catherine Chang. (March 2003). Programming methodology for biologically-inspired self-assembling systems. In *AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality*.

[22] C. W. Reynolds. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34.

[23] L. Spector, J. Klein, C. Perry, and M. Feinstein. (2005). Emergence of collective behavior in evolving populations of flying agents. *Genetic Programming and Evolvable Machines*, 6(1):111–125.

[24] Larry A. Taber. (2004). *Nonlinear Theory of Elasticity: Applications in Biomechanics*. World Scientific, Singapore.

[25] C. M. Topaz, A. L. Bertozzi, and M. A. Lewis. (2006). A nonlocal continuum model for biological aggregation. *Bulletin of Mathematical Biology*, 68:1601–1623.