Path Creation by Continuous Flocking as an Example of a Morphogenetic Programming Language

Bruce J MacLennan
maclennan@utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_elecpubs

Part of the Artificial Intelligence and Robotics Commons, and the Programming Languages and Compilers Commons

Recommended Citation
MacLennan, Bruce J, "Path Creation by Continuous Flocking as an Example of a Morphogenetic Programming Language" (2018). Faculty Publications and Other Works -- EECS. https://trace.tennessee.edu/utk_elecpubs/24

This Report is brought to you for free and open access by the Engineering -- Faculty Publications and Other Works at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Faculty Publications and Other Works -- EECS by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.
Path Creation by Continuous Flocking as an Example of a Morphogenetic Programming Language

Bruce J. MacLennan*

Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
web.eecs.utk.edu/~mclennan

June 10, 2018

Abstract

Artificial morphogenesis uses processes inspired by embryology to control massive swarms of robots to assemble complex physical structures. First, we use an example morphogenetic program to illustrate a prototype implementation of morphgen, a morphogenetic programming language. The syntax and semantics are described informally as 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.

*This report may be used for any non-profit purpose provided that the source is credited.
1 Introduction

Artificial morphogenesis uses processes inspired by embryology to control massive swarms of robots to assemble complex physical structures. In this report we use two-dimensional continuous flocking to create paths from an origin to a destination while avoiding obstacles. It is a simple example of artificial morphogenesis and massive swarm robotics, used here 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 a number of previous papers [1, 2, 3, 4, 5, 6, 7, 8, 9]. In particular, our approach to continuous flocking is described in [6, 7, 9].

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.

The morphogenetic programming language, tentatively named morphgen, adheres fairly closely to the mathematical notation used in publications. The prototype implementation illustrated in this report is implemented by a syntax macroprocessor (tentatively named “synmac”) [11]. 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 Octave. Although synmac is quite flexible, it does not include a full parser, and so some syntactic concessions must be made for 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 (p. 28).

2 Description of a Morphogenetic Program

The complete morphogenetic program is shown in Appendix A (p. 26). The first two (somewhat obscure) lines are directives to the syntax macroprocessor:

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

The first line directs the macroprocessor to load the definitions for the 2D version of the morphgen language (“.smac” is the extension we have adopted for syntax macros). The second line directs it to allow underscores and decimal points in tokens, so that they are permitted in names and numbers.

Much of the rest of 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:

```
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 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 \(200 \times 200\) 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, the `goal_material`, a `morphogen`, the `swarm`, and the `path_material`. The definition of the morphogen substance (lines 21–29) is perhaps most illustrative:

```
substance morphogen:
  scalar field A
  behavior:
    let d_A = 0.03
    let k_G = 100
    let tau_A = 100
    let tau_P = 0.2
    \( \text{D} A = d_A \nabla^2 A + k_G [G(1-A)] - A/\tau_A - [P*A]/\tau_P \)
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. The first four lines after `behavior` simply define constants. The last line, which begins with \(\text{D}\), is a `change equation`, which can be interpreted ambiguously as a partial differential equation or temporal finite difference equation. The programming language statement

```
\( \text{D} A = d_A \nabla^2 A + k_G [G(1-A)] - A/\tau_A - [P*A]/\tau_P \)
```

is an approximation to the usual morphogenetic programming notation, which we have used in previous publications:

\[
D A = d_A \nabla^2 A + k_G [G(1-A)] - A/\tau_A - [P*A]/\tau_P. \tag{1}
\]

This 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. However, multiplication of scalar fields or of scalar fields by vector fields must be surrounded by square brackets; for example, “[P*A]” is a multiplication of scalar fields. Similarly the quotient of a vector field by a scalar field and powers of scalar fields must be surrounded by brackets.

Equation 1 can be explained as follows. The first term $d_A \nabla^2 A$ describes the diffusion of the morphogen $A$. The second term $k_G G(1 - A)$ describes production in the goal region $G = 1$ of morphogen $A$ up to saturation ($A = 1$). The third term $-A/\tau_A$ describes decay of the morphogen. The final term $-PA/\tau_P$ describes rapid decay or absorption of the morphogen where there is path material ($P > 0$), which keeps the swarm away from obstacles. In the absence of obstacles, the steady-state concentration of morphogen at a distance $r$ from the goal ($G = 1$) is given by

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

(i.e., the characteristic length constant is $\sqrt{d_A \tau_A}$).

Also simple is the path substance (lines 51-56), which represents material being laid down by the swarm as well as by previous paths, which are obstacles to be avoided:

```
substance path_material:
  scalar field P
  behavior:
    let k_P = 30
    D P = [t > 5] k_P * [C * (1 - P)]
end
```

The concentration of path material is represented by the scalar $P$ field. The factor $k_P * [C * (1 - P)]$, 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 conditional factor $[t > 5]$ has the value 1 when time $t > 5$ and 0 otherwise. This has the effect of suppressing path generation for the first five time units in order to let the morphogen gradient stabilize before path material is deposited.

The goal material is static, so it has a very simple definition (lines 15-19). On the other hand, the swarm substance (lines 31-49) is the most complicated; 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 since it lays down the path material, as just explained. The behavior part of the substance definition begins with four parameter definitions:

\[
\begin{align*}
\text{behavior:} & \\
& \text{let } v = 1  \quad /*  base  swarm  speed  */ \\
& \text{let } \lambda = 0.01 \\
& \text{let } \epsilon = 1e^{-100} \\
& \text{let } k_W = 0.1
\end{align*}
\]

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

The next three lines define the vector field that directs the swarm’s movement:

\[
\begin{align*}
\text{let } & U = \text{del} \ A \\
\text{let } & S = \|U\| \\
\text{let } & V = \left[ \frac{(v \cdot U)}{(S+\epsilon)} \right] - \lambda \text{del}[(C-1)^2] + [k_W \text{DW}^2]
\end{align*}
\]

The first line gives a name to the morphogen gradient (\( U = \nabla A \)). Since the gradient may vary greatly in magnitude, we normalize it. To this end, the scalar field \( S \) is defined \( S = \|U\| \). The last line defines the directive vector field:

\[
V = \frac{vU}{(S+\epsilon)} - \lambda \nabla (C-1)^2 + k_W \text{DW}^2. \tag{2}
\]

The first term is the speed times the normalized gradient, that is, \( vU/(S+\epsilon) \), or “[\((v \cdot U) / (S+\epsilon)]” in the programming language. The second term, \( -\lambda \nabla (C-1)^2 \), written “−\( \lambda \text{del}[(C-1)^2] \),” controls the density of the swarm to keep it compact but not too dense. This term (controlled by \( \lambda \)) directs motion in a direction that minimizes \( |C-1| \) and therefore strives to keep the density near 1. The last term, “[\( k_W \text{DW}^2 \)],” introduces some randomness into the swarm’s movement to help break symmetry and avoid deadlocks. The notation “[\( k_W \text{DW}^2 \)]” represents \( k_W \text{DW}^2 \), a two-dimensional normally-distributed random vector \([1, 2, 3, 5]\). The final line in the swarm substance behavior describes its movement:
The change in concentration $C$ is given by the negative divergence of the flux,
$- \text{div}(C \mathbf{V}) = - \nabla \cdot C \mathbf{V}$, but movement is suppressed for the first five time units
so that the morphogen gradient can stabilize.

After the substance definitions comes the initialization of the various bodies. The
simplest is the Goal definition (lines 58–60), which defines the small region from which
the morphogen diffuses and which the swarm seeks:

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

The Goal is near the far limit $y = 0.925$ on the $x$ axis.

The swarm Cohort (lines 69–71) is initially at the origin of the path, which is on
the $x$ axis near the opposite side of the space $(y = -0.925)$:

```plaintext
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, we define several pre-existing
concentrations of path material to represent them (lines 61–66); since this is a 2D
simulation, they are simply circular regions where $P = 1$.

```plaintext
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
```

Regions of any field that are not initialized are by default zero.
Figure 1: Example plots of swarm density ($t = 5.5$). The swarm is splitting into two sub-swarms to go around an obstacle.

The final block in the program (lines 73–79) defines the visualization options:

```plaintext
visualization:
  display interval = 0.05
  display final P as colors limits (0, 0.5)
  display running C as colors limits (0, 0.5)
  display final P as mesh
end
```

Fields can be displayed either at the end of the simulation, indicated by the keyword `final`, or while the simulation is running, 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., Fig. 1a); `mesh` displays a scalar field as a 3D surface (e.g., Fig. 1b), and `contours` displays a scalar field as a contour map (e.g., Fig. 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., Fig. 2). A running display can be made into an mp4 movie with a command such as this:

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

This creates a movie file called “PathFormation” from a running display of $P$ displayed as colors.

### 3 Continuous Flocking 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 the algorithm and to further illustrate the morphgen2D language. It’s apparent that the algorithm involves
many parameters, and so it’s necessary to explore their effect on the outcome so they can be adjusted to achieve any desired end. Moreover, as in any algorithm, we may entertain different ways of accomplishing various operations, some of which work and others don’t. 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 Fig. 3. Nominal parameters for the simulations are summarized in Table 1.

Figure 4 shows the effect of $\lambda$, 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 is moving entirely under the influence of the morphogen gradient. Figures 4b to 4d show the paths created with successively larger values of $\lambda$, 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 morphogen language includes visualization commands to report various numerical condition numbers. In this case we used the statement

\begin{verbatim}
    report Courant number for V
\end{verbatim}

Figure 2: Vector field $\mathbf{V}$ ($\lambda = 0, t = 6$) displayed as quivers (50 × 50 grid).
Figure 3: Final concentration of path material.

<table>
<thead>
<tr>
<th>Param.</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>6.75</td>
<td>duration</td>
</tr>
<tr>
<td>$\Delta s$</td>
<td>0.01</td>
<td>spatial resolution</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>0.001</td>
<td>temporal resolution</td>
</tr>
<tr>
<td>$d_A$</td>
<td>0.03</td>
<td>attractant diffusion constant</td>
</tr>
<tr>
<td>$\tau_A$</td>
<td>100</td>
<td>attractant decay time constant</td>
</tr>
<tr>
<td>$k_G$</td>
<td>100</td>
<td>attractant release rate from goal substance</td>
</tr>
<tr>
<td>$\tau_P$</td>
<td>0.2</td>
<td>attractant absorption time constant</td>
</tr>
<tr>
<td>$v$</td>
<td>1</td>
<td>base swarm speed</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.03</td>
<td>importance of swarm density</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$10^{-100}$</td>
<td>minimum attractant gradient magnitude</td>
</tr>
<tr>
<td>$k_W$</td>
<td>0.3</td>
<td>amount of random motion</td>
</tr>
<tr>
<td>$k_P$</td>
<td>30</td>
<td>path deposition rate</td>
</tr>
</tbody>
</table>

Table 1: Nominal Parameter Values
Figure 4: Effects of $\lambda$ on path structure ($k_P = 30, t = 6.75$, color limits = [0, 0.5]).
to display the Courant number

$$C_r = \left( \max_{x,y} |V_x(x,y)| + |V_y(x,y)| \right) \Delta t / \Delta s$$

(for time step size $\Delta t$ and mesh spacing $\Delta s$) at every display interval. 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. 2), a large density gradient can result in a high velocity, causing numerical instability and possibly physical problems. We can compare the magnitude of the morphogen gradient, constrained to the rate $v$, with the magnitude of the density-driven gradient, $\| \nabla (C - 1)^2 \|$:

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

Therefore, the relative magnitudes of the morphogen and density components of the velocity are $v$ and $2\lambda \sqrt{\nabla^2 C}$ respectively, and large density gradients can 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 U and V vector fields as follows:

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

$$\frac{dA(r)}{dr} = \frac{d}{dr}k_G \exp \left( -\frac{r}{\sqrt{d_A r_A}} \right) = -\frac{k_G}{\sqrt{d_A r_A}} \exp \left( -\frac{r}{\sqrt{d_A r_A}} \right).$$

Therefore the gradient varies with distance from the goal, and so the relative contributions to the velocity of the morphogen and density gradients 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.

Another approach (suggested by Allen McBride) is to normalize 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 $W = \nabla(C-1)^2$ be the density gradient and $T = ||W||$ be its magnitude. Then,

$$V = v[(1 - \lambda)U/(S + \epsilon) - \lambda W/(T + \epsilon) + k_W DW^2].$$

(3)

This is accomplished by the following morphgen code:

| 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]) |

(The notation “...” allows an equation to be continued in order to improve readability.) The $\lambda > 0$ parameter does control the density, causing the swarm to break up into small compact groups and lay down paths of relatively constant width (Fig. 6). This also delays arrival at the destination, since the effective rate following the morphogen is $v(1 - \lambda)$.

Figure 7 illustrates the effect of the $\epsilon$ parameter (“eps” in the program), which is primarily to guard against division by zero when normalizing the morphogen gradient (using the original V computation, Eq. 2). The nominal value is $10^{-100}$ and values up to about $10^{-8}$ lead to very similar patterns (Fig. 7a). The path patterns become more diffuse with larger values (e.g., Fig. 7b for $\epsilon = 10^{-6}$). The explanation is that the magnitude of the morphogen gradient is small compared to $\epsilon$, and therefore the morphogen-directed speed is less than $v$. For these large $\epsilon$ values the swarm takes longer to move from the origin and therefore longer to reach the goal (e.g., $T = 10$ vs. nominal $T = 6.75$). The delayed start is presumably because the morphogen gradient is still increasing in comparison with $\epsilon$.  

12
Figure 6: Path densities resulting from normalizing morphogen and density gradients before combination ($T = 14$, color limits $= [0, 1]$).
Figure 7: Effect of eps or $\epsilon$ parameter on path density ($\lambda = 0.01, \tau_P = 0.2, k_P = 30$).

Figure 8 exhibits the effect of temporal and spatial resolution on the simulation (original $V$ computation, Eq. 2). The plots on the right have higher resolution ($\Delta s = 5 \times 10^{-3}, \Delta t = 2.5 \times 10^{-4}$) than those on the left ($\Delta s = 10^{-2}, \Delta t = 10^{-3}$), which have the nominal resolutions used in most of these experiments. The $\lambda = 0$ simulations are virtually identical. For $\lambda = 0.005$ and 0.03 the lower and higher resolutions produce qualitatively similar path densities, but the lower resolution plots are not simply blurred versions of the higher resolution plots. This may be a consequence of the higher velocities resulting from $\lambda > 0$, for the morphogen-generated velocity is limited to $v$, but the density-generated velocity is not limited.

Equation 2 for $V$ (p. 5) includes a random element $k_W D W^2$, the purpose of which is to break a symmetry that otherwise leads to unrealistic results. The problem 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. Therefore, instead of going around the obstacles, a small part of the swarm “tunnels” through it. (The simulation does not model the fact that the obstacles are solid and therefore impenetrable.). This can be seen in Fig. 9a, 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 (Figs. 9b–9d). 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. 1 (p. 3). Smaller values of the time constant $\tau_P$ lead to quicker elimination of attractant,
Figure 8: Effects of better temporal and spatial resolution on path density. Plots on the right have higher spatial and temporal resolution than those on the left. From top, plots have $\lambda = 0, 0.005, 0.03$. 

(a) $\lambda = 0, \Delta s = 0.01, \Delta t = 0.001$
(b) $\lambda = 0, \Delta s = 0.005, \Delta t = 0.00025$
(c) $\lambda = 0.005, \Delta s = 0.01, \Delta t = 0.001$
(d) $\lambda = 0.005, \Delta s = 0.005, \Delta t = 0.00025$
(e) $\lambda = 0.03, \Delta s = 0.01, \Delta t = 0.001$
(f) $\lambda = 0.03, \Delta s = 0.005, \Delta t = 0.00025$
Figure 9: Effect of randomness on path formation. Sufficient randomness eliminates “tunneling” through obstacles.
Figure 10: Margins around obstacles resulting from various time constants $\tau_P$ for attractant elimination ($\lambda = 0.02$).
which does indeed lead to larger margins. Figure 10 shows path densities resulting from several different values of $\tau_P$, 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 attractant from all directions equally, and that this generally steers the swarm away from them. This can be seen in Fig. 11, 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. (Recall that the equation for $\nabla P$ causes it to saturate at $P = 1$.) This can be seen in Fig. 12a. 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 $-k_C G C$ term to the $\nabla C$ equation:

\[
\text{let } k_C = 100 /* \text{swarm absorption rate} */ \\
\mathbf{D} \ C = [t > 5] -\nabla [C\ast V] - k_C \ast [G\ast C]
\]

Notice that in Fig. 12a the path density at the goal has saturated at its maximum value $P = 1$, whereas in Fig. 12b 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., Figs. 3, 4, 12). Since the paths represent bundles of fibers, some variation in density across the
Figure 12: 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 $P \approx 0.8$.

Figure 13: Use of autocatalysis and decay to control path density.

(a) $\theta_P = 0.2$, no decay  
(b) $\theta_P = 0.3$, with decay ($t_P = 1$)
width of a path is unproblematic, but we expect consistent density 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 over a specified threshold. This is accomplished by adding a term \([P > \theta_P]P(1 - P)\) to the \(DP\) equation. By itself, this leaves the low density path material in the environment (Fig. 13a), but it can be eliminated by a decay term \(-P/t_P\) operative for below-threshold densities:

\[
DP = [t > 5](k_P C(1 - P) + a_P[P > \theta_P]P(1 - P) - [P \leq \theta_P]P/t_P).
\]

(4)

The program code to accomplish this is:

```
let k_P = 30
let a_P = 20       /* autocatalytic rate */
let theta_P = 0.3  /* autocatalytic threshold */
let t_P = 1        /* path decay */
let Q = a_P*[P > theta_P] [P*(1-P)] - [P <= theta_P]P/t_P
DP = [t >5](k_P*[C*(1-P)] + Q)
```

Figure 13b 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 \(DP\) equation produces well-defined paths, but the density is variable (Fig. 14a). This can be avoided by combining quorum sensing in path deposition with autocatalysis and decay of the path material (Eq. 4) to obtain:

\[
DP = [t > 5](k_P[C > \theta_C]C(1 - P) + a_P[P > \theta_P]P(1 - P) - [P \leq \theta_P]P/t_P).
\]

In above-quorum regions, the path material will increase to saturation through autocatalysis (Figs. 14b–14d). Lower quorum thresholds \(\theta_C\) 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.
Figure 14: Path formation with swarm quorum sensing.
<table>
<thead>
<tr>
<th>Param.</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>10</td>
<td>duration</td>
</tr>
<tr>
<td>$\Delta s$</td>
<td>0.01</td>
<td>spatial resolution</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>0.001</td>
<td>temporal resolution</td>
</tr>
<tr>
<td>$d_A$</td>
<td>0.03</td>
<td>attractant diffusion constant</td>
</tr>
<tr>
<td>$\tau_A$</td>
<td>100</td>
<td>attractant decay time constant</td>
</tr>
<tr>
<td>$k_G$</td>
<td>100</td>
<td>attractant release rate from goal substance</td>
</tr>
<tr>
<td>$\tau_P$</td>
<td>0.1</td>
<td>attractant absorption time constant</td>
</tr>
<tr>
<td>$v$</td>
<td>1</td>
<td>base swarm speed</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.5</td>
<td>importance of swarm density</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>$10^{-100}$</td>
<td>minimum attractant gradient magnitude</td>
</tr>
<tr>
<td>$k_W$</td>
<td>0.7</td>
<td>amount of random motion</td>
</tr>
<tr>
<td>$k_P$</td>
<td>30</td>
<td>path deposition rate</td>
</tr>
<tr>
<td>$a_P$</td>
<td>20</td>
<td>path autocatalysis rate</td>
</tr>
<tr>
<td>$\theta_P$</td>
<td>0.3</td>
<td>path autocatalysis threshold</td>
</tr>
<tr>
<td>$t_P$</td>
<td>1</td>
<td>path decay time constant</td>
</tr>
<tr>
<td>$k_C$</td>
<td>100</td>
<td>swarm absorption rate</td>
</tr>
</tbody>
</table>

Table 2: Revised Parameter Values

Figure 15: Simulations based on revised parameter values (Table 2). The smaller $\tau_P$ eliminates tunneling.
4 Revised Nominal Parameters

Drawing on the preceding experiments, we can collect parameters that give good results (Table 2). By default, we use prenormalization of the morphogen and density-control gradients (Eq. 3, p. 12), and to promote uniform path density, we use autocatalysis with decay (Eq. 4, p. 20), but not quorum sensing. Figure 15 shows two simulations: Fig. 15a has $\tau_P = 0.2$, which has tunneling through the first obstacle, and Fig. 15b 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 (Fig. 16). (These simulations were also run at higher resolution: $\Delta s = 0.005, \Delta t = 0.0005$.) Figure 16a 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, Fig. 16b 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 (Fig. 16c). Figure 16d shows that this can be filled in by lowering the autocatalysis threshold $\theta_P$ 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 17 shows the path created by a simulation with the parameters shown, which used the original version of the velocity equation:

$$\text{let } V = \left( \frac{v*U}{S+\text{eps}} \right) - \lambda * \text{del} \left( (C-1)^2 \right)$$

A later report will explore the 3D simulation in more detail.
Figure 16: Simulations at higher resolution ($\Delta s = 0.005, \Delta t = 0.0005$) with different obstacles, origin (lower right), and destination (upper left).
Figure 17: 3D simulation of path formation \((d_A = 0.03, k_G = 100, \tau_A = 100, \tau_P = 0.2, v = 1, \lambda = 0.03, \epsilon = 10^{-100}, \tau_C = 0.01, k_P = 30, T = 6.5, \Delta s = 0.01, \Delta t = 0.001)\).
A 2D Continuous Flocking Program

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

// Continuous Flocking

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 goal_material:
  scalar field G
  behavior:
    D G = 0 /* G field is fixed */
end

substance morphogen:
  scalar field A
  behavior:
    let d_A = 0.03
    let k_G = 100
    let tau_A = 100
    let tau_P = 0.2
    D A = d_A*\delta^2 A + k_G*[G*(1-A)] - A/tau_A - [P*A]/tau_P
end

substance swarm:
  scalar fields:
    C /* swarm concentration */
    S /* magnitude of morphogen gradient */
end
  vector fields:
    U /* morphogen gradient */
    V /* swarm velocity */
end
  behavior:
let v = 1 /* base swarm speed */
let lambda = 0.03
let eps = 1e-100
let k_W = 0.1
let U = del A
let S = ||U||
let V = [(v*U)/(S+eps)] - lambda*del[(C-1)^2] + [k_W D W^2]
D C = [t>5] -div [C*V]
end

substance path_material:
  scalar field P
  behavior:
    let k_P = 30
    D P = [t>5] k_P * [C * (1 - P)]
end

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

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

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

visualization:
  display interval = 0.05
  display final P as colors limits(0, 0.5)
  display running C as colors limits(0, 0.5)
  display final P as mesh
  report Courant number for V
end

end program
B  morphgen2D Syntax

Notation: Square brackets surround optional items; curly braces group items. (When
used as terminal symbols, they are in boldface.) Superscript * means zero or more
occurrences, superscript + means one or more repetitions. ⟨comment⟩s can appear
anywhere whitespace is allowed (generally, between tokens). Line continuation 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 :
   ⟨equation⟩*
end
⟨variable⟩ ::= scalar field ⟨name⟩⟨newline⟩
   | vector field ⟨name⟩⟨newline⟩
   | scalar fields : ⟨newline⟩⟨name list⟩ end
   | vector fields : ⟨newline⟩⟨name list⟩ end
⟨name list⟩ ::= { ⟨name⟩⟨newline⟩ }+ 
⟨equation⟩ ::= D ⟨name⟩ [+|−] = ⟨expr⟩⟨newline⟩
   | let ⟨name⟩ = ⟨expr⟩⟨newline⟩
   | ⟨log params⟩
⟨expr⟩ ::= ⟨primitive⟩[(⟨operator⟩⟨primitive⟩)]* 
⟨operator⟩ ::= + | − | * | / | ^
\(\text{⟨primitive⟩} ::= \langle\text{name}\rangle | \langle\text{num}\rangle | (\langle\text{expr}\rangle) | \langle\text{special}\rangle\)

\(\text{⟨special⟩} ::= \text{del} \langle\text{primitive}\rangle\)
| \(\text{del}'2 \langle\text{primitive}\rangle\)
| \(\text{div} \langle\text{primitive}\rangle\)
| \(||\langle\text{expr}\rangle||\)
| \([\langle\text{primitive}\rangle > \langle\text{primitive}\rangle] \langle\text{primitive}\rangle\)
| \([\langle\text{primitive}\rangle >= \langle\text{primitive}\rangle] \langle\text{primitive}\rangle\)
| \([\langle\text{primitive}\rangle < \langle\text{primitive}\rangle] \langle\text{primitive}\rangle\)
| \([\langle\text{primitive}\rangle <= \langle\text{primitive}\rangle] \langle\text{primitive}\rangle\)
| \([\langle\text{primitive}\rangle * \langle\text{primitive}\rangle]\)
| \([\langle\text{primitive}\rangle / \langle\text{primitive}\rangle]\)
| \([\langle\text{primitive}\rangle ^ \langle\text{primitive}\rangle]\)
| \([\langle\text{primitive}\rangle \text{DW} ^ \langle\text{primitive}\rangle]\)

\(\text{⟨body⟩} ::= \text{body} \langle\text{name}\rangle \text{of} \langle\text{name}\rangle : \langle\text{definition}\rangle* \text{end}\)

\(\text{⟨definition⟩} ::= \text{for} \langle\text{region}\rangle : \langle\text{name}\rangle = \langle\text{expr}\rangle \langle\text{newline}\rangle\)
| \(\text{for} \langle\text{region}\rangle : \langle\text{newline}\rangle \langle\text{init}\rangle* \text{end}\)

\(\text{⟨region⟩} ::= \langle\text{expr}\rangle < \langle\text{name}\rangle < \langle\text{expr}\rangle, \langle\text{expr}\rangle < \langle\text{name}\rangle < \langle\text{expr}\rangle\)
| \((\langle\text{name}\rangle, \langle\text{name}\rangle) \text{within} \langle\text{expr}\rangle \text{of} ((\langle\text{expr}\rangle), (\langle\text{expr}\rangle))\)

\(\text{⟨init⟩} ::= \langle\text{name}\rangle = \langle\text{expr}\rangle \langle\text{newline}\rangle\)

\(\text{⟨visualization⟩} ::= \text{visualization} \langle\text{vis command}\rangle + \text{end}\)

\(\text{⟨vis command⟩} ::= \text{display} \langle\text{time}\rangle \langle\text{primitive}\rangle \text{as} \langle\text{kind}\rangle\)
| \(\text{display running code} \langle\text{target code}\rangle \text{end code}\)
| \(\text{make movie} \langle\text{filename}\rangle \text{of} \langle\text{primitive}\rangle \text{as} \langle\text{kind}\rangle \langle\text{newline}\rangle\)
| \(\text{record parameters in} \langle\text{filename}\rangle \langle\text{newline}\rangle\)
| \(\langle\text{stability report}\rangle\)

\(\text{⟨time⟩} ::= \text{running} | \text{final}\)

\(\text{⟨kind⟩} ::= \{\text{mesh} | \text{contours} | \text{colors}\} \text{[limits} ((\langle\text{expr}\rangle), (\langle\text{expr}\rangle))]]\)
| \(\text{quivers} \text{[grid} ((\langle\text{expr}\rangle), (\langle\text{expr}\rangle))]]\)

\(\text{⟨stability report⟩} ::= \text{report diffusion number for} \langle\text{primitive}\rangle\)
| \(\text{report Courant number for} \langle\text{primitive}\rangle\)
| \(\text{report Peclet number for} \langle\text{primitive}\rangle \text{and} \langle\text{primitive}\rangle\)

\(\text{⟨log params⟩} ::= \text{log params} \langle\text{name}\rangle [, \langle\text{name}\rangle]* \langle\text{newline}\rangle\)

\(\text{⟨name⟩} ::= \langle\text{letter}\rangle [ \langle\text{letter}\rangle | \langle\text{digit}\rangle | \_ ]*\)

\(\text{⟨num⟩} ::= \[-\](\langle\text{digit}\rangle[.\langle\text{digit}\rangle]*)\)

\(\text{⟨comment⟩} ::= /\* \langle\text{characters}\rangle */ | // \langle\text{characters}\rangle \langle\text{newline}\rangle\)
References


