Pacific Graphics 2022
N. Umetani, E. Vouga, and C. Wojtan
(Guest Editors)

Volume 40 (2021), Number 7

MoMaS: Mold Manifold Simulation for real-time procedural texturing

F. Maggioli[1], R. Marin[1], S. Melzi[2] and E. Rodolà[1]

[1]Sapienza - University of Rome, Italy
[2]Università di Milano Bicocca, Italy

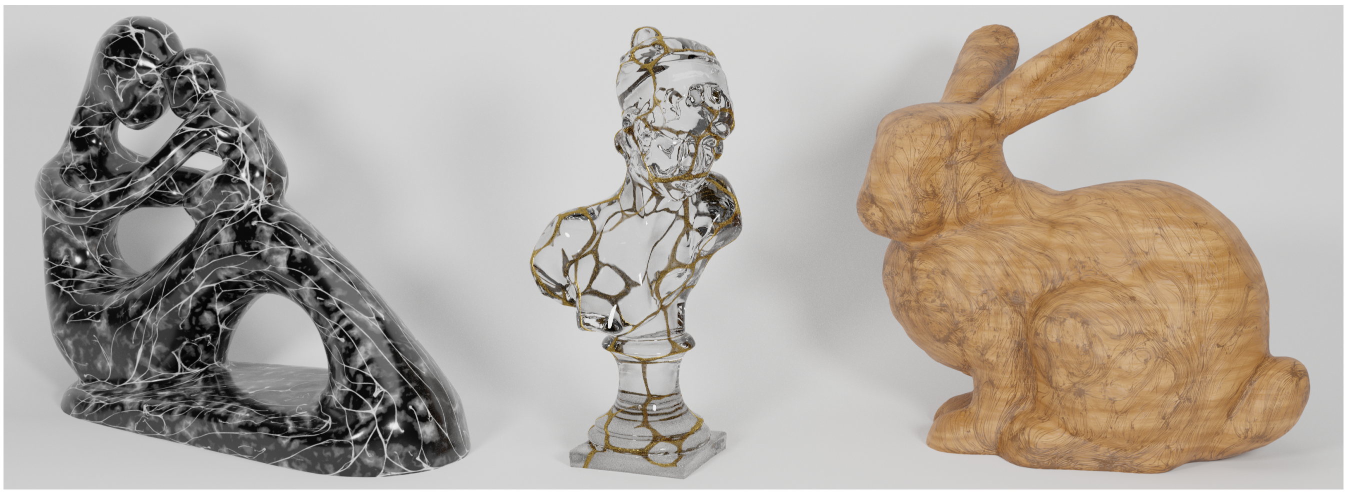


Figure 1: *Different materials generated from the simulation of our slime approach at a certain frame: marble and wooden grain (left and right respectively), or golden web on a glass structure (middle). Look at supplementary video for an animated visualization of the pattern evolution.*

Abstract

The slime mold algorithm has recently been under the spotlight thanks to its compelling properties studied across many disciplines like biology, computation theory, and artificial intelligence. However, existing implementations act only on planar surfaces, and no adaptation to arbitrary surfaces is available. Inspired by this gap, we propose a novel characterization of the mold algorithm to work on arbitrary curved surfaces. Our algorithm is easily parallelizable on GPUs and allows to model the evolution of millions of agents in real-time over surface meshes with several thousand triangles, while keeping the simplicity proper of the slime paradigm. We perform a comprehensive set of experiments, providing insights on stability, behavior, and sensibility to various design choices. We characterize a broad collection of behaviors with a limited set of controllable and interpretable parameters, enabling a novel family of heterogeneous and high-quality procedural textures. The appearance and complexity of these patterns are well-suited to diverse materials and scopes, and we add another layer of generalization by allowing different mold species to compete and interact in parallel.

Keywords: Procedural texturing, animated texture, slime mould, GPU algorithm

CCS Concepts
• **Computing methodologies** → **Texturing;** • **Theory of computation** → *Parallel algorithms;* • **Mathematics of computing** → *Geometric topology;*

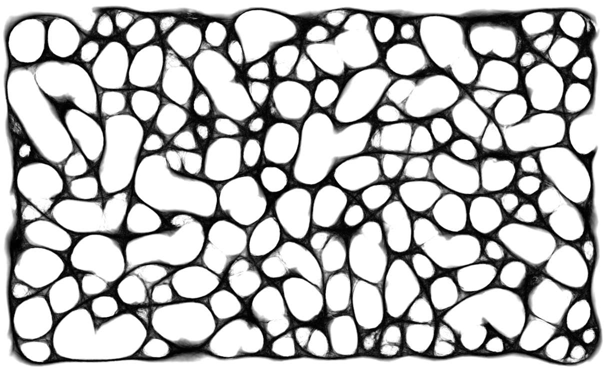


Figure 2: *Visualization of the pheromone trace in a slime mold simulation. The simulation involves 1M agents and takes place on a bounded flat region.*

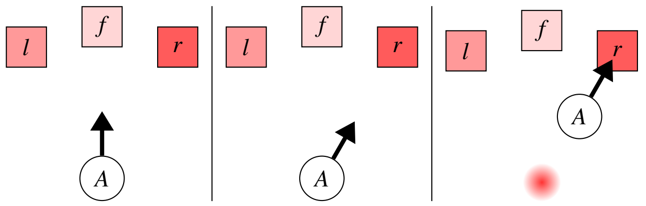


Figure 3: *The behavior of a slime agent A. The agent samples from three sensors in front of it (left l, right r, and forward f) and determines the next direction by choosing the one with highest pheromone concentration (darker red). When leaving a location, the agent releases pheromone.*

1. Introduction

In different disciplines, the interest in biological evolutive systems has grown in recent years. Biology and chemistry scholars and, more recently also bioinformatics, artificial intelligence, and computation theory researchers are focusing on such complex systems. More specifically, systems that can produce a consistent global behavior by a few local rules are compelling due to their simplicity but powerful expressive capabilities. Slime mold systems are included in this family [Jon10]. Biological mold organisms propagate following pheromones attraction – from a computer scientist's perspective, perfectly fitting the divide-and-conquer paradigm. These properties have motivated researchers to simulate their evolution, exploiting modern computational capabilities to explore and investigate their properties. Among the available solutions, no method offers an implementation on arbitrary surfaces to the best of our knowledge. We believe this is not due to a lack of interest in non-Euclidean domains, but rather because these dynamic simulations on curved surfaces require a thorough background in differential geometry, not common in the biological community.

From a Computer Graphics perspective, producing complex patterns over a surface belongs to the procedural texturing domain. Despite the increasing availability of computational power and advances in geometry processing, texture design is still a human-centered and time-consuming task for the most. The growing amount and quality of geometrical assets urge the generation of realistic synthetic textures, while guaranteeing the quality and efficiency required by the entertainment industry.

In this paper, we present a high-quality, real-time implementation of a slime mold algorithm for arbitrary surfaces with potentially millions of agents running in parallel. We design it led by simplicity, efficiency, and generality principles. We introduce only the necessary technicalities, accepting approximations that keep the method simple but are not visually harmful to the final quality. Our method entirely takes place over the surface and offers a set of parameters that artists can easily interpret, so as to produce various distinguishable patterns. These can be used for modeling different materials, as shown in Figure 1. We further show how to constrain the mold evolution to specific regions by defining repulsion / attraction areas.

Our contribution can be summarized as follows:

- we provide the first slime mold algorithm for surfaces, with an analysis of its behavior; we show that it is predictable, and respects the expected properties of this kind of organisms;
- we define a new family of patterns for procedural texturing that is interpretable, controllable and admitting path constraints in the pattern evolution;
- we release a light-speed implementation that scales well at different texture resolutions, number of agents, and mesh resolution, opening to real-time video applications and massive texture generation.

1.1. Related work and background

Slime mold simulation The evolution of biological patterns is a vast research area in biology, and it often produces interesting visualizations that can also be used in movies and generative art. A famous example of pattern formation is the evolution of the *Physarum Polycephalum*, also known as slime mold. The algorithm first presented in [Jon10] produces complex patterns like the one shown in Figure 2. In recent years, the slime mold simulation has become popular because of its large area of application, which covers cognition, optimization, computation and machine learning [VCM*18, BA17, EIZO20, LCW*20, ABMC*21]. The slime mold algorithm is also starting to be used in generative art [MF21] and visualization techniques [EBPF20]. The algorithm for simulating slime molds has proven to be so valuable that some researchers are starting to integrate it in larger pipelines to achieve compelling results [ZGS20, ABCM20].

Slime simulation falls under the category of systems that can produce complex behaviors from a set of simple rules, like cellular automata simulation [CD98] and reaction-diffusion systems [WK91]. The rules of the system can be summarized as follows:

- A set of agents lives in a closed space where they can move freely;
- Each agent releases a pheromone trace that diffuses and evaporates over time;
- Each agent tries to follow the pheromone trace in its field of view.

Figure 3 summarizes how agents act during an iteration. Each agent samples three regions in front of it at an angle (forward, left, or right sensor). The region with the most pheromone traces determines the steering direction of the agent. While moving, the agent releases more pheromone, which in turn diffuses locally and evaporates over time. The simulation can be tuned according to several parameters affecting the agents and the ambient.

Despite the increasing interest in this argument, the research in this direction has been limited to tuning and optimizing the slime mold algorithm, or to its application in different areas.

Procedural texturing The main works in this context are devoted to procedural generation of height maps for landscapes [LN03, Ols04, Par14], patterns for planar surfaces [WK91], 3D noise functions [CD05, Har01] or mixes of other textures [EMP*03]. Other works have proposed to use procedural texturing for vector field visualization [BK08] or the representation of complex repetitive geometries [Ney95], but still, these works are limited to flat domains. Texture synthesis on surfaces has been first addressed in [WL01], and during the years the research greatly advanced; see [WLKT09] for an extensive survey. However, the works in this area mainly exploit example-based methods, adapting them to curved domains [LH06], or they are limited to simple patterns [KCPS15]. The only few exceptions are simulation based approaches [Tur91, Sta03], which makes them similar in spirit to our method. Only recently, some works have shown procedural generation of complex patterns directly on surfaces [NPP21, FPSG21], but these works are limited to recursive structures or repetitive patterns and do not explore other possibilities. Another work following a similar fashion is [MNPP21], where the authors propose a method for drawing Bézier curves on manifold meshes. Still, such curves require a set of input points from the user, which is unfeasible for creating large and complex patterns.

Riemannian geometry In the continuous setting, we represent a 3D shape as a compact Riemannian surface $\mathcal{M}$ embedded in $\mathbb{R}^3$. We briefly recap some mathematical preliminaries, and refer to [Cha06, dC16] for further details. We denote $T_x\mathcal{M}$ the tangent plane at a point $x \in \mathcal{M}$. The tangent plane gives a linear approximation of the surface $\mathcal{M}$ locally around the point x. The *tangent bundle* $T\mathcal{M} := \cup_{x\in\mathcal{M}} T_x\mathcal{M}$ is the disjoint union of all the tangent planes. On each tangent plane we define an inner product $g_x(\cdot,\cdot)$, uniquely determined by a 2×2 matrix $\mathbf{g}_x$ called *metric tensor*. The metric tensor determines the length distortion of a vector v in $\mathbb{R}^2$ when mapped to $T_x\mathcal{M}$. The actual length of v in $T_x\mathcal{M}$ is given by $\|v\|_{\mathbf{g}} = \sqrt{v^\top \mathbf{g} v}$.

A curve on $\mathcal{M}$ consists of a diffeomorphism $\gamma : (-1,1) \in \mathbb{R} \to \mathcal{M}$. Given a local chart φ, the differential $\frac{\mathrm{d}}{\mathrm{d}t}(\varphi\circ\gamma) \in T_x\mathcal{M}$ represents the direction of the infinitesimal movement of a point (an agent in this paper) along the curve γ at $x = \gamma(0)$. Thus, the tangent plane contains the direction of the infinitesimal movement of an agent moving on the surface. The differential of a curve is independent on the local chart [dC16, MSNN01]. In the discrete setting, we approximate the manifold $\mathcal{M}$ as a triangle mesh with n vertices, whose coordinates are stored in a matrix $V \in \mathbb{R}^{n\times 3}$. The edges between them define the triangular faces, the union of which approximates the continuous surface. Tangent planes, tangent bundles, and differential constructions have a discrete definition for triangular meshes; we refer to [CdGDS13] for a complete description.

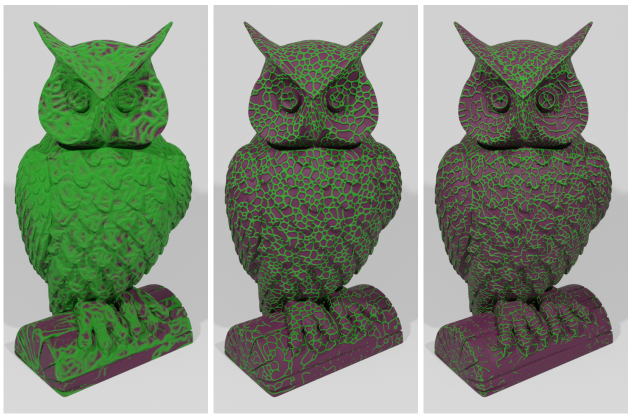
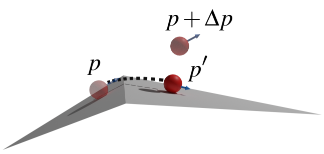


Figure 4: *Effect of sub-step subdivision on the slime mold simulation. We use the same simulation parameters in all three cases, but vary the number of sub-steps. Left to right: single step, 10 sub-steps, 100 sub-steps.*

2. Method

In our setting, an agent $\mathbf{A}$ is a particle on $\mathcal{M}$ that moves over the surface following slime simulation rules. We require that our method: i) works on general meshes and topologies; ii) allows $\mathbf{A}$ to move on arbitrary paths in real time; iii) guarantees numerical stability. This section provides a detailed description of how we define our method to achieve such desiderata.

2.1. Movement over the surface



The motion of a particle over a surface is a well-known problem. A widely used solution is triangle unfolding [SSC19, SGC21], but to keep the implementation simple we decide to approximate the path by moving the particle in small steps and reprojecting it on the surface.

Given the initial position p of $\mathbf{A}$, let t be the triangle that contains p, and let Δp be the vector encoding a forward step. We split Δp into smaller sub-steps; at each sub-step, we re-project $\mathbf{A}$'s position onto the surface. To ease the projection computation, we consider only the triangle t' that is the nearest to $p+\Delta p$ and incident to t (see inset). This movement parcellation limits inconsistency produced by edge crossing and guarantees we cannot cover more than one triangle at a time. Alternatively, one could compute exactly when an edge has been crossed and split the movement perfectly between the incident triangles. However, we argue that our approach does not require any check, and the approximation produced is negligible. Our sub-step strategy is crucial, as can be seen in Figure 4; the leftmost owl does not use any sub-step subdivision, resulting in an unrecognizable pattern. While increasing the number of sub-steps produces a more precise approximation of the motion, we stress that the resulting path does not converge to a geodesic. In

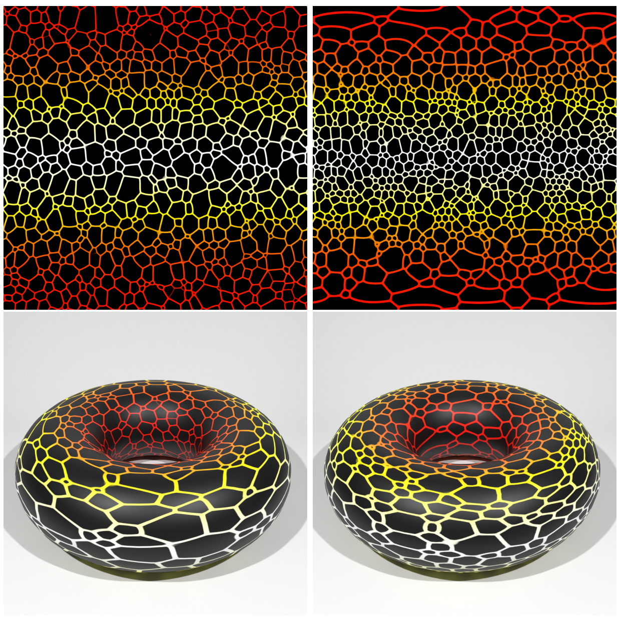


Figure 5: *Comparison between two pheromone traces on a torus with and without length rescaling. On the left, the agent movements in texture space are of the same size, producing a pattern with inconsistent lengths in 3D. On the right, lengths in texture space are rescaled according to the metric, generating a more uniform pattern in 3D. We represent the metric $\|\cdot\|_{\mathbf{g}}$ as a colormap growing from dark red to white.*

fact, slime agents move along a constant direction in the parametric space, which does not necessarily coincide as moving along a geodesic.

2.2. Length rescaling

In practice, we operate in texture space and represent the position of each agent by two coordinates, and its direction as a single scalar (i.e., the steering angle). The pheromone is encoded as color. If the agent does not cross the border of a triangle during a step, the change in position is easily computed by summing two 2-dimensional vectors. However, this choice also requires taking into account the metric distortion induced by the UV mapping.

To guarantee that a vector a in texture space has uniform 3D length on the mesh domain, we divide it by the norm $\|a\|_{\mathbf{g}} = \sqrt{a^\top \mathbf{g} a}$, where $\mathbf{g}(t)$ is the discrete metric tensor at triangle t. Since the latter depends entirely on the triangle t, it can be pre-computed for all triangles at initialization. In Figure 5, we depict an example of this correction in texture space and 3D space.

2.3. Mold evolution

Our mold evolution process follows the spirit of [Jon10]; we summarize it in Algorithm 1. We pre-compute the set of adjacent triangles and the metric tensor at each triangle (Lines 2-6). Then, we simulate a certain number of steps. At each step, all the agents sense

Algorithm 1 Slime mold on surfaces.

```
1:  procedure MoMaS($\mathcal{M}$, $h_f$, $\Delta h$)
2:      for all triangle $t$ do
3:          // Pre-compute adjacency and metric tensor
4:          Compute Adj($t$)
5:          Compute $\mathbf{g}(t)$
6:      end for
7:      $h \leftarrow 0$
8:      while $h < h_f$ do
9:          for all agent $a$ do
10:             // Sense the pheromone trace
11:             for $\theta \in \{-\vartheta_s,\ 0,\ \vartheta_s\}$ do
12:                 $\Delta s \leftarrow (\cos(a.\theta + \theta),\ \sin(a.\theta + \theta))$
13:                 $(s_\theta, -, -) \leftarrow$ TangentStep($a.p$, $\delta_s \Delta s$, $a.t$)
14:                 $P_\theta \leftarrow$ Pheromone($s_\theta$)
15:             end for
16:             // Determine next direction and move the agent
17:             $\theta^* \leftarrow \arg\max_{\theta \in \{-\vartheta_s,\ 0,\ \vartheta_s\}} \{P_\theta\}$
18:             $a.\theta \leftarrow a.\theta + \vartheta_a \theta^*$
19:             $\Delta p \leftarrow (\cos(a.\theta),\ \sin(a.\theta))$
20:             $(a.p, a.t, a.\theta) \leftarrow$ TangentStep($a.p$, $\delta_a \Delta p$, $a.t$)
21:             // Release pheromone
22:             Pheromone($a.p$) $\leftarrow 1$
23:         end for
24:         // Apply global pheromone diffusion and evaporation
25:         Blur the texture Pheromone
26:         Pheromone $\leftarrow \max(\text{Pheromone} - \varepsilon_d, 0)$
27:         $h \leftarrow h + \Delta h$
28:     end while
29: end procedure
```

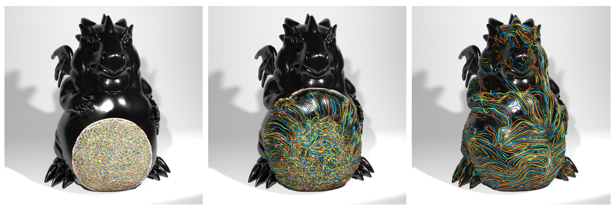


Figure 6: *Evolution of a 3-species slime mold over a surface. Agents start from an initial region and diffuse over the entire mesh.*

three positions in front of them by sampling the texture Pheromone (Lines 11-15). The sensor placement depends on two parameters: the sensor distance δ_s and the sensor angle ϑ_s. The location with the highest quantity of pheromone attracts the agent (Line 17). The agent then turns to that direction with turn speed ϑ_a and moves along the new direction with velocity δ_a (Lines 18 to 20). Some randomness can be added to the agent's steering for adding extra dynamism and random changes to the pattern. Here, TangentStep is any procedure that moves a point over the tangent space and returns the final position, the triangle and the direction aligned with the movement. For additional details about our implementation, we

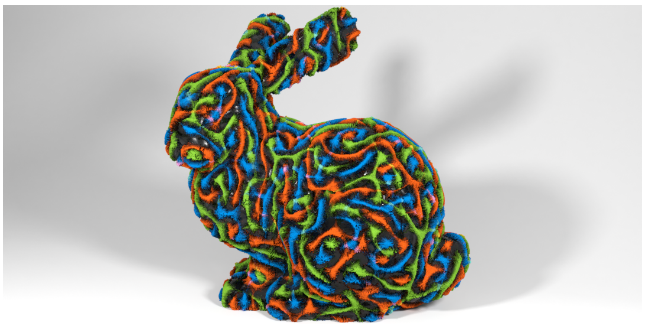


Figure 7: *An example of simulation with multiple species, repelling each other. The pheromone trace of each species is stored in a different channel of the texture and is represented with a different color.*

remand to the supplementary materials. and the source code[†]. Once the agents finish releasing their pheromone in their new position on the texture Pheromone (Line 22), we apply a blurring algorithm to diffuse the pheromone and remove a small quantity that evaporates with velocity ε_d (Lines 25 and 26). The parameters introduced in the last lines characterize the behavior and the obtained pattern, and we analyze them in detail in Section 3.3.

As a final note, to keep our implementation simple and GPU-friendly, we accomplish the blurring step (Line 25) with a standard Gaussian blur directly on the texture. While this can in principle generate visible seams, it allows us to handle very high-resolution textures in real time, and the artifacts on texture seams are not visually noticeable as we demonstrate in our experiments.

2.4. Agent implementation

Agents are represented by a data structure with the following fields:

- p: position of the agent in texture space;
- θ: direction angle in texture space;
- t: triangle containing point p.

Each agent contains only four values, making the representation memory-efficient. Despite the triangle being a piece of redundant information (it could be inferred by p), computing it at each step becomes unfeasible in terms of performance.

Since the agents act independently, we implement their behavior on GPU, achieving real-time performance on the simulation.

2.5. Multiple Species, Obstacles and Attractors

The slime mold algorithm allows for further generalization, including multiple species of agents; see Figure 6. Each different species releases a different type of pheromone, attracting agents of the same species and repelling others. To implement this mechanics, we need to distinguish between pheromone types and define ‘attractive’ and ‘repelling’ pheromone. Since we encode the pheromone trace in a texture, we assign a channel of the texture to each species. When sampling for pheromone at a given cell, an agent adds the pheromone from the channel of its species and subtracts the pheromone from the channel of other species. This way, agents from the same species will tend to aggregate and isolate from other species. In Figure 7, we visualize an example of this behavior.

[†] https://github.com/filthynobleman/slime-manifold

Parameter		Property		Tested Range
Movement Speed	δ_a	Pattern Scale	↑	$[1.0, 2.0]$
Turn Speed	ϑ_a	Stabilization	↓	$[10.0, 50.0]$
Vision Distance	δ_s	Cell Formation	↑	$[0.4, 2.0]$
Vision Angle	ϑ_s	Thickness	↑	$[10°, 50°]$
Evaporation Rate	ε_d	Clustering	↓	$[0.2, 1.0]$

Table 1: *Each parameter is associated with a specific property of the generated pattern most affected by its variation. The arrow indicates whether the parameter affects the property positively (green up arrow) or negatively (red down arrow). The last column shows the range we tested for the parameter.*

Similarly, we exploit attractive / repelling pheromone to model obstacles and attractors. An obstacle is a region R_O that agents must avoid, whereas an attractor is a region R_A that agents must reach and never leave. While defining such areas for general particle dynamic systems would be complex, we can do this easily by treating obstacles and attractors as particular pheromones that never diffuse and evaporate. For the obstacles, we assign to the region R_O an infinitely large amount of pheromone that repels all the species (*i.e.* negative pheromone). This trick guarantees that an agent prefers any other direction rather than entering into R_O. For the attractors, we define an amount of attractive pheromone superior to the number of species but not infinite. Since the pheromone of species has values in $[0, 1]$, this always guarantees attraction, but still makes the agents able to produce patterns inside the target region.

3. Results

Before passing to a quantitative and qualitative evaluation, we provide a description of the key parameters of our algorithm.

Parameters Among the parameters, five of them have a higher impact on the produced pattern in our experiments. More in detail:

δ_a: movement speed (*i.e.* how fast agents move);
ϑ_a: turning speed (*i.e.* how fast agents change direction);
δ_s: distance of vision (*i.e.* how far the sensor is placed);
ϑ_s: angle of vision (*i.e.* how widely sensors are placed);
ε_d: evaporation rate (*i.e.* how fast the pheromone decays).

In Table 1, for each parameter, we show the tested range of values and the property that is mainly affected by each of them.

3.1. Mold Simulation

The generic evolution of the pattern follows these steps:

1. the agents are initialized randomly;

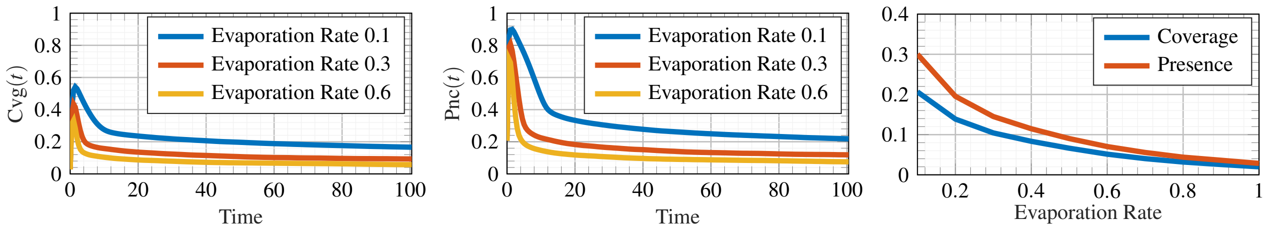


Figure 8: *Evolution of the pheromone coverage (a) and presence (b) during a simulation of 100 seconds. After the initial peak, the agents stabilize on the pattern family and slowly vary the resulting pattern. Different decay rates shift the curve, without changing the evolution. The pheromone coverage and presence after 70 seconds of simulation vary as the decay rate changes (c). The slower the evaporation, the higher the coverage and presence. We performed these experiments on the shape on the left of Figure 1.*

2. the agents start to move freely, covering large portions of the surface;
3. the agents rapidly aggregate, and the properties of the pattern family show up;
4. the pattern changes slowly but continuously, without changing family.

These steps are common to all configurations and are not affected by the parameters, even if changing the parameters can produce small changes in the timing.

To give a quantitative idea of the pattern distribution and identify when the process reaches each step, we studied how much surface is covered by pheromone over time. More formally, we define the *pheromone coverage* Cvg(t) and the *pheromone presence* Pnc(t), as:

$$\begin{aligned}\mathrm{Cvg}(t) &= \frac{1}{\mathrm{Full}}\int_{\mathcal{M}} \mathrm{Pheromone}_t(x)\,\mathrm{d}x \\ \mathrm{Pnc}(t) &= \frac{1}{\mathrm{Full}}\int_{\mathcal{M}} \lceil \mathrm{Pheromone}_t(x) \rceil\,\mathrm{d}x\end{aligned} \tag{1}$$

where Pheromone$_t(x)$ is the quantity of pheromone at point x and at time t and takes values in $[0,1]$, and $\lceil\cdot\rceil$ is the ceiling operator. The values are normalized with respect to the total coverage Full $= \int_{\mathcal{M}} \mathrm{d}x$.

Figures 8a and 8b show the typical evolution of pheromone coverage and presence during a simulation, with different values of evaporation rate. After the random initialization, the agents start to move freely, and they cover large portions of the mesh, reaching a peak. As the pattern is formed, the coverage decreases. The stable line after the peak represents the slow evolution of the pattern. Biological simulations also exhibit a similar behavior, showing our method acts as expected [PRAD21,MZB*21]. The evaporation rate affects pheromone coverage and presence, as shown in Figure 8c. The other parameters do not affect the pheromone coverage and have a marginal impact on the pheromone presence. We refer to Figure 1 in the supplementary materials to support this claim.

Attractors and obstacles. In Figure 9, we show a simulation with attractors and obstacles. The ability to solve mazes is a well-studied property of organisms like the *Physarum Polycephalum* [Ada12, Nak01], and we tested our method in this context. The agents are initialized in the red region and expand toward the attractive region (in green), avoiding the walls. In our implementation, we use a three-channel texture to define the starting, repelling, and attractive regions, each associated with a different channel. In contrast, in Figure 6 we show a 3-species mold evolution over a surface with no obstacles or attractors, thus leaving the agents completely free.

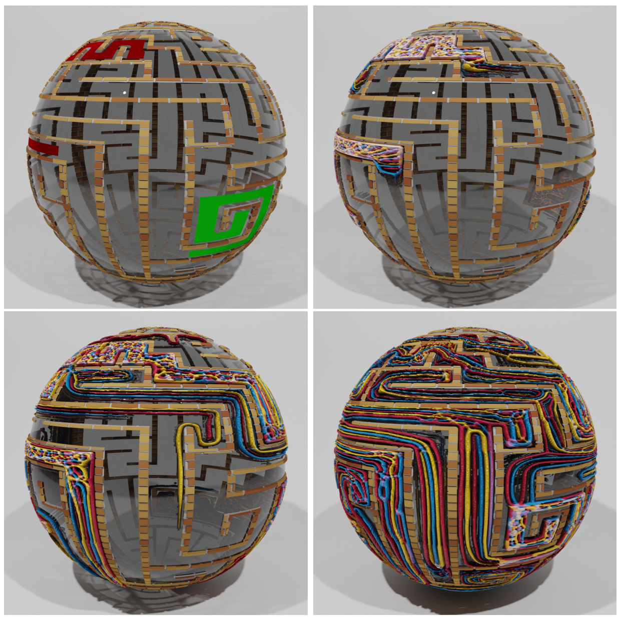


Figure 9: *A sphere is decorated with a texture representing a maze (top-left frame). The red region is the starting zone, and the green region is the attractor. Three species of slime are initialized in the starting region (top-right) and evolve over the maze (bottom-left). When the agents reach the attractor, they start forming patterns inside and never leave the attracting region (bottom-right).*

3.2. Performance

We run our simulations on a fixed set of ten meshes (see supplementary material) and under different conditions. All the experiments have been carried out on a RTX 2080 Ti.

tex. size	avg. tpf	max. tpf	min. tpf
2048	8.473	37.277	7.362
4096	9.503	37.693	8.268
8192	12.120	39.039	10.263
16384	23.715	43.552	16.220

Table 2: *Time per frame (in milliseconds) at different texture resolutions. Simulations are ran using 10 sub-steps and involve 1M agents.*

num. steps	avg. tpf	max. tpf	min. tpf
1	12.070	37.204	10.020
10	13.428	38.632	11.087
100	47.828	61.519	44.756
1000	423.136	488.078	411.836

Table 3: *Time per frame (in milliseconds) at different steps per frame. Simulations are ran on a* 8192×8192 *texture and involve 1M agents.*

num. agents	avg. tpf	max. tpf	min. tpf
1M	12.053	39.522	10.214
2M	19.112	37.205	17.306
3M	26.303	41.857	24.006
4M	33.510	53.710	31.114

Table 4: *Time per frame (in milliseconds) at different number of agents. Simulations are ran on a* 8192×8192 *texture and use 10 sub-steps.*

Texture resolution. Table 2 summarizes the average, maximum, and minimum time per frame using textures at different resolutions. We run the simulations for 900 frames with the same settings. In particular, we use ten sub-steps in the tangent space. Despite the increase in time per frame, results show that our algorithm achieves real-time performance at very high resolutions.

Number of sub-steps. To show the impact of our movement parcellation policy, in Table 3, we show the average, maximum and minimum time per frame at different number of sub-steps. We set the texture resolution to 8192×8192. As for Table 2, simulations are done for 900 frames, fixing the other simulation parameters. These results show that one or ten steps perform similarly. Subdividing by 100 or 1000 steps, the frame rate significantly drops. We conjecture that up to 100 steps, the GPU threads warm-up cover a significant portion of the frame time. This dependency on the number of sub-steps relates the execution time to the mesh resolution. For denser meshes, the number of triangles crossed by a single step increases, together with the number of sub-steps required to keep the result consistent. This, in turn, increases the execution time.

Number of agents. Finally, we are interested in studying the efficiency of our implementation while varying the number of agents. Table 4 summarizes our results over 900 frames. The increase in the average time per frame is almost linear (about 7 ms at each increment of one million agents), showing our approach scales well with the number of agents.

3.3. Families of patterns

Our parameters allow to generate different types of pattern. The movement speed controls the pattern scale; as we increase the velocity of the agents, the pattern becomes uniform, and local details tend to disappear. Figure 10 shows this variation together with the variation in the evaporation rate. Here the values of the parameters are linearly interpolated inside the ranges shown in Table 1. This parameter weighs on the tendency of agents to form clusters. When the evaporation rate is low, the pheromone stays on for longer, and agents can travel by longer distances following it. As the evaporation rate increases, the agents tend to aggregate only with other agents already near them, and thus they tend to form small clusters. The placement of sensors also affects the formation of cells and the thickness of the trails; we refer to Section 2 of the supplementary for a qualitative study of these dependencies.

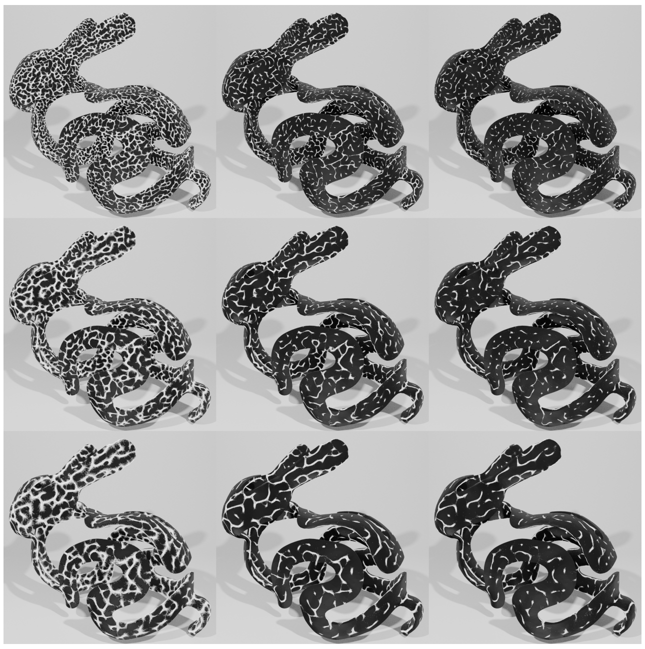


Figure 10: *Changes in the resulting pattern when varying the movement speed and the evaporation rate. The change in movement speed scales up the pattern along each column, making it wider. As the evaporation rate increases along each row, the agents tend to form shorter chains and collapse into local clusters.*

3.4. Evolutive procedural texturing

The creation of complex patterns on meshes is a time-consuming task for texture artists. For many applications, such as landscape generation or tile synthesis, procedural texturing has become very useful and effective [MSC12, Ols04, LN03, Par14]. Nevertheless, most of the literature is devoted to the generation of complex textures on planar domains, and only a few works deal with general manifold meshes [NPP21, MNPP21, FPSG21]. The ability to generate a broader range of complex patterns that follow the shape of a mesh is of fundamental importance in generating a large variety of contents. Moreover, [Jon10] shows that it is possible to obtain a variety of really different patterns by simply tuning the simulation parameters, as shown in Figure 11. For the base patterns used to generate these rendering, we refer to Figure 4 of the supplementary material.

Visualizing the evolution of patterns and effects using textures

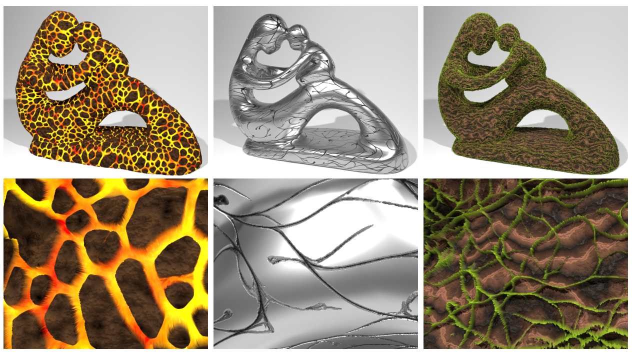


Figure 11: *Frames from simulations with the same mesh and initial conditions but with different simulation parameters. The produced patterns have been used as composition mask and details maps for more complex materials. The second row shows a close-up detail of the surface.*

has been extensively used for time-dependent vector fields and scalar functions in 2D domains [Ney03, BK08]. However, animating materials and textures over surfaces is currently limited to very simple shaders and texture changes. The evolution of slime mold organisms on a surface effectively animates a coloration of the mesh at the texture level. In Figures 1 and 11 we show how to exploit our method to simulate properties like wood grain, metal incisions, or lava rivers.

3.5. Limitations

Our method inherits the main limitations of texture-based approaches. For example, dealing with meshes with many small triangles may cause a performance drop due to the high resolution required to model pixels and the step subdivision. However, this does not impact the correctness of our method, and we expect this situation to occur only for contexts where real-time performance is less relevant.

Moreover, if adjacent triangles in a mesh generate an angle smaller than 90 degrees can produce inconsistencies in reprojecting the agents. In this case, triangle unfolding can be a more desirable option for moving the agent over the surface. For this reason, we implemented the algorithm so that TANGENTSTEP can be viewed as a black-box routine and replaced according to the user needs.

4. Conclusions

We presented a generalization of the slime mold algorithm to surface meshes and an analysis of its behavior and controllability with simulation parameters. The algorithm produces complex patterns on triangle meshes, behaving similarly to real biological entities. We discussed the applicability of our method to computer graphics tasks like procedural texturing. Finally, we discussed our GPU implementation and its applicability to real-time texture animation.

We consider studying how molds evolve on surfaces with different geometrical properties as an interesting future direction. This analysis could open to other applications, *e.g.* extrapolating geometry descriptors or intrinsic properties of the surface from the mold's distribution.

Acknowledgment

We thank professor Marco Tarini for its invaluable feedback on an early version of the manuscript. This work is supported by the ERC grant no. 802554 (SPECGEO), the SAPIENZA BE-FOR-ERC 2020 Grant (NONLINFMAPS), and an Alexander von Humboldt Foundation Research Fellowship.

References

[ABCM20] ABDEL-BASSET M., CHANG V., MOHAMED R.: HSMA_WOA: A hybrid novel Slime mould algorithm with whale optimization algorithm for tackling the image segmentation problem of chest X-ray images. *Applied Soft Computing 95* (2020), 106642. URL: https://www.sciencedirect.com/science/article/pii/S1568494620305809, doi:https://doi.org/10.1016/j.asoc.2020.106642. 2

[ABMC*21] ABDEL-BASSET M., MOHAMED R., CHAKRABORTTY R. K., RYAN M. J., MIRJALILI S.: An efficient binary slime mould algorithm integrated with a novel attacking-feeding strategy for feature selection. *Computers & Industrial Engineering 153* (2021), 107078. URL: https://www.sciencedirect.com/science/article/pii/S0360835220307488, doi:https://doi.org/10.1016/j.cie.2020.107078. 2

[Ada12] ADAMATZKY A.: Slime mold solves maze in one pass, assisted by gradient of chemo-attractants. *IEEE transactions on nanobioscience 11*, 2 (2012), 131–134. 6

[BA17] BURGIN M., ADAMATZKY A.: Structural machines and slime mould computation. *International Journal of General Systems 46*, 3 (2017), 201–224. URL: https://doi.org/10.1080/03081079.2017.1300585, arXiv:https://doi.org/10.1080/03081079.2017.1300585, doi:10.1080/03081079.2017.1300585. 2

[BK08] BELCHER J., KOLECI C.: Using animated textures to visualize electromagnetic fields and energy flow. *arXiv preprint arXiv:0802.4034* (2008). 3, 8

[CD98] CHOPARD B., DROZ M.: *Cellular automata*, vol. 1. Springer, 1998. 2

[CD05] COOK R. L., DEROSE T.: Wavelet noise. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 803–811. 3

[CdGDS13] CRANE K., DE GOES F., DESBRUN M., SCHRÖDER P.: Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses* (New York, NY, USA, 2013), SIGGRAPH '13, ACM. 3

[Cha06] CHAVEL I.: *Riemannian geometry: a modern introduction*, vol. 98. Cambridge university press, 2006. 3

[dC16] DO CARMO M.: *Differential Geometry of Curves and Surfaces: Revised and Updated Second Edition*. Dover Books on Mathematics. Dover Publications, 2016. URL: https://books.google.it/books?id=gg2xDQAAQBAJ. 3

[EBPF20] ELEK O., BURCHETT J. N., PROCHASKA J. X., FORBES A. G.: Polyphorm: structural analysis of cosmological datasets via interactive physarum polycephalum visualization. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2020), 806–816. 2

[EIZO20] EKINCI S., IZCI D., ZEYNELGIL H. L., ORENC S.: An application of slime mould algorithm for optimizing parameters of power system stabilizer. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)* (2020), pp. 1–5. doi:10.1109/ISMSIT50672.2020.9254597. 2

[EMP*03] Ebert D. S., Musgrave F. K., Peachey D., Perlin K., Worley S.: *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003. 3

[FPSG21] Fanni F. A., Pellacini F., Scateni R., Giachetti A.: Pavel: Decorative patterns with packed volumetric elements, 2021. arXiv:2102.01029. 3, 7

[Har01] Hart J. C.: Perlin noise pixel shaders. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), pp. 87–94. 3

[Jon10] Jones J.: Characteristics of pattern formation and evolution in approximations of physarum transport networks. *Artificial Life 16* (2010), 127–153. URL: https://uwe-repository.worktribe.com/output/980579, doi:10.1162/artl.2010.16.2.16202. 2, 4, 7

[KCPS15] Knöppel F., Crane K., Pinkall U., Schröder P.: Stripe patterns on surfaces. *ACM Trans. Graph. 34* (2015). 3

[LCW*20] Li S., Chen H., Wang M., Heidari A. A., Mirjalili S.: Slime mould algorithm: A new method for stochastic optimization. *Future Generation Computer Systems 111* (2020), 300–323. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19320941, doi:https://doi.org/10.1016/j.future.2020.03.055. 2

[LH06] Lefebvre S., Hoppe H.: Appearance-space texture synthesis. *ACM Trans. Graph. 25*, 3 (jul 2006), 541–548. URL: https://doi.org/10.1145/1141911.1141921, doi:10.1145/1141911.1141921. 3

[LN03] Lefebvre S., Neyret F.: Pattern based procedural textures. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2003), I3D '03, Association for Computing Machinery, p. 203–212. URL: https://doi.org/10.1145/641480.641518, doi:10.1145/641480.641518. 3, 7

[MF21] McGraw T., Ferdousi B.: Red versus blue: Slime mold civil war. In *SIGGRAPH Asia 2021 Posters* (New York, NY, USA, 2021), SA '21 Posters, Association for Computing Machinery. URL: https://doi.org/10.1145/3476124.3488619, doi:10.1145/3476124.3488619. 2

[MNPP21] Mancinelli C., Nazzaro G., Pellacini F., Puppo E.: b/surf: Interactive bézier splines on surfaces. *arXiv preprint arXiv:2102.05921* (2021). 3, 7

[MSC12] Maung D., Shi Y., Crawfis R.: Procedural textures using tilings with perlin noise. In *2012 17th International Conference on Computer Games (CGAMES)* (2012), IEEE, pp. 60–65. 7

[MSNN01] Morita S., Society A. M., Nagase T., Nomizu K.: *Geometry of Differential Forms*. Iwanami series in modern mathematics. American Mathematical Society, 2001. URL: https://books.google.it/books?id=5N33Of2RzjsC. 3

[MZB*21] Marbach S., Ziethen N., Bastin L., Baeuerle F., Alim K.: Network architecture determines vein fate during spontaneous reorganization, with a time delay. *bioRxiv* (2021). 6

[Nak01] Nakagaki T.: Smart behavior of true slime mold in a labyrinth. *Research in Microbiology 152*, 9 (2001), 767–770. 6

[Ney95] Neyret F.: Animated texels. In *Computer Animation and Simulation '95* (Vienna, 1995), Terzopoulos D., Thalmann D., (Eds.), Springer Vienna, pp. 97–103. 3

[Ney03] Neyret F.: Advected Textures. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (San diego, United States, July 2003), Breen D., Lin M., (Eds.), Eurographics Association, pp. 147–153. URL: https://hal.inria.fr/inria-00537472. 8

[NPP21] Nazzaro G., Puppo E., Pellacini F.: Geotangle: Interactive design of geodesic tangle patterns on surfaces. *ACM Trans. Graph. 41*, 2 (nov 2021). URL: https://doi.org/10.1145/3487909, doi:10.1145/3487909. 3, 7

[Ols04] Olsen J.: Realtime procedural terrain generation. 3, 7

[Par14] Parberry I.: Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques 3*, 1 (2014). 3, 7

[PRAD21] Patino-Ramirez F., Arson C., Dussutour A.: Substrate and cell fusion influence on slime mold network dynamics. *Scientific reports 11*, 1 (2021), 1–20. 6

[SGC21] Sharp N., Gillespie M., Crane K.: Geometry processing with intrinsic triangulations. In *ACM SIGGRAPH 2021 Courses* (New York, NY, USA, 2021), SIGGRAPH '21, Association for Computing Machinery. URL: https://doi.org/10.1145/3450508.3464592, doi:10.1145/3450508.3464592. 3

[SSC19] Sharp N., Soliman Y., Crane K.: Navigating intrinsic triangulations. *ACM Trans. Graph. 38*, 4 (jul 2019). URL: https://doi.org/10.1145/3306346.3322979, doi:10.1145/3306346.3322979. 3

[Sta03] Stam J.: Flows on surfaces of arbitrary topology. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, Association for Computing Machinery, p. 724–731. URL: https://doi.org/10.1145/1201775.882338, doi:10.1145/1201775.882338. 3

[Tur91] Turk G.: Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph. 25*, 4 (jul 1991), 289–298. URL: https://doi.org/10.1145/127719.122749, doi:10.1145/127719.122749. 3

[VCM*18] Vallverdú J., Castro O., Mayne R., Talanov M., Levin M., Baluška F., Gunji Y., Dussutour A., Zenil H., Adamatzky A.: Slime mould: The fundamental mechanisms of biological cognition. *Biosystems 165* (2018), 57–70. URL: https://www.sciencedirect.com/science/article/pii/S0303264717304574, doi:https://doi.org/10.1016/j.biosystems.2017.12.011. 2

[WK91] Witkin A., Kass M.: Reaction-diffusion textures. In *Proceedings of the 18th annual conference on computer graphics and interactive techniques* (1991), pp. 299–308. 2, 3

[WL01] Wei L.-Y., Levoy M.: Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 355–360. 3

[WLKT09] Wie L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the Art in Example-based Texture Synthesis. In *Eurographics 2009 - State of the Art Reports* (2009), Pauly M., Greiner G., (Eds.), The Eurographics Association. doi:10.2312/egst.20091063. 3

[ZGS20] Zhao J., Gao Z.-M., Sun W.: The improved slime mould algorithm with levy flight. *Journal of Physics: Conference Series 1617* (aug 2020), 012033. URL: https://doi.org/10.1088/1742-6596/1617/1/012033, doi:10.1088/1742-6596/1617/1/012033. 2