MoMaS: Mold Manifold Simulation for real-time procedural texturing
Supplementary Materials

F. Maggiotti1, R. Marin1, S. Melzi2 and E. Rodolà1

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

In this document we provide additional information and more detailed insights on the experimental results presented in the main paper. Moreover, we give additional details about our implementation of the algorithm for moving a particle over a surface.

1. Particle Motion Over a Surface

Algorithm 1 summarizes the entire process. The procedure TANGENTSUBSTEP is given a point \( p \) in texture space and the triangle \( t \) where \( p \) lies. It computes the movement over the tangent space of the mesh along some vector \( \Delta p \). The vector is still defined in texture space, and thus must be rescaled according to the local metric tensor \( g(t) \) (Line 3). The final position after the movement is converted to barycentric coordinates to determine if the border of the triangle has been crossed (Lines 5-10). The adjacent triangle \( w \) closest to the final point is selected and the point is projected on it (Lines 12 and 13). Finally, the coordinates are converted to 2D via barycentric coordinates (Lines 15 and 16). To handle meshes with boundaries, it is sufficient to identify if the agent crossed a boundary edge (i.e. there is no adjacent triangle on that edge). If this is the case, we make the agent bounce on the edge.

The vectors \( n_t \) and \( \iota_t \) are, respectively, the normal and the barycenter of a triangle \( \tau \), while Adj(\( \tau \)) is the set of its adjacent triangles. The matrices \( T_\tau \) and \( L_\tau \) are defined for each triangle \( \tau \) as

\[
T_\tau = \begin{pmatrix} r_{\tau,1} - r_{\tau,3} & r_{\tau,2} - r_{\tau,3} \end{pmatrix} \in \mathbb{R}^{2 \times 2}
\]

\[
L_\tau = \begin{pmatrix} v_{\tau,1} & v_{\tau,2} & v_{\tau,3} \end{pmatrix} \in \mathbb{R}^{3 \times 3}
\]

(1)

where \( r_{\tau,j} \) are the coordinates of the vertices of \( \tau \) in texture space and \( v_{\tau,j} \) are the coordinates of the vertices of \( \tau \) in 3D space. The matrix \( T_\tau \) is the conversion matrix from barycentric to edge coordinates and is such that \( T_\tau \lambda' = p - r_{\tau,3} \), given that \( \lambda' \) are the first two components of the barycentric coordinates of \( p \). Matrix \( L_\tau \) is the conversion matrix from barycentric to Cartesian coordinates. Usually, it is defined as a rectangular matrix in \( \mathbb{R}^{4 \times 3} \), since it must have a row of ones in order to ensure the point is inside the triangle. In our case, we need to allow points to move outside triangles, and thus we remove that row. As a side effect, we can move back and forth between Cartesian and barycentric coordinates by inverting \( L_\tau \), which saves much computation with respect to the classic area-based method.

The procedure TANGENTSTEP iterates the process multiple times to continuously determine the new position and direction in texture space, as well as the current triangle. The number of sub-steps \( N \) is some fixed constant depending on the mesh resolution and it is used to determine the length of the sub-step (Line 3). At each iteration, the algorithm moves the agent by a small step (Line 6) and then uses a slightly larger step to determine the next direction (Line 8). The direction is then normalized and rescaled to a proper length (Line 9). The call to TANGENTSTEP also has the job of determining the next direction and the triangle of the next position to pass them to the next iteration.

2. Area Coverage and Parameters Tuning

Figure 1 accompanies Figure 8 from the main paper. Differently from the evaporation rate, the movement speed of the agents and their angle of vision seems to not affect the area coverage. Even if changing these parameters produces very different patterns, the portion of surface area covered by the pheromone trace does not vary. The plots for the agents’ turn speed and the sensors’ distance behave similarly.

Figure 10 in the main paper shows an interpolation between simulation parameters, in order to give an insight on how a single parameter affects the behavior of the resulting pattern. Figure 2 and Figure 3 show a more detailed overview of the parameters’ interpolations. Figure 2 accompanies Figure 10 from the main paper, whereas Figure 3 analyzes the effect of changing the vision distance and angle parameters. In both cases, the parameters are linearly interpolated using the ranges in Table 1 from the main manuscript. The turning speed of the agents does not seem to affect the pattern over the surface. Instead, we observed a correlation between turning speed and how fast the pattern changes during its evolution. Moreover, we notice that changing the movement speed and the evaporation rate has a higher visual impact than changing the sensors’ parameters. A small change in the latter is visually more subtle.
Algorithm 1 Step in the tangent space of a mesh.

1: \textbf{procedure} TANGENTSSTEP\((p, \Delta p, t)\)
2: \hspace{1em} // Rescale the displacement vector
3: \quad \Delta p \leftarrow \frac{\Delta p}{\sqrt{\langle \Delta p \rangle \cdot g(t) \Delta p}}
4: \hspace{1em} // Convert to barycentric coordinates
5: \quad q \leftarrow p + \Delta p
6: \quad \lambda \leftarrow \sum_{i=1}^{3} \left( q - r_{i} \right)
7: \quad \lambda_{3} \leftarrow 1 - \lambda_{1} - \lambda_{2}
8: \hspace{1em} \text{if} \lambda \text{ is inside } t \text{ then}
9: \hspace{2em} \text{return} \(t, p + \Delta p\)
10: \hspace{1em} \text{end if}
11: \hspace{1em} // Search for the nearest adjacent triangle
12: \quad w \leftarrow \arg \min_{\tau \in \text{Adj}(t)} \{|\langle n_{\tau}, q - c_{\tau} \rangle|\}
13: \quad q \leftarrow q - \langle n_{w}, q - c_{w} \rangle n_{w}
14: \hspace{1em} // Return final texture coordinates and the new triangle
15: \quad \lambda \leftarrow L_{w}^{-1} q
16: \quad \text{return} \(w, \lambda_{1} r_{w,1} + \lambda_{2} r_{w,2} + \lambda_{3} r_{w,3}\)
17: \textbf{end procedure}

18: \textbf{procedure} TANGENTSSTEP\((p, \Delta p, t)\)
19: \quad p' \leftarrow p
20: \quad \Delta p' \leftarrow \frac{\Delta p}{N}
21: \hspace{1em} \text{for} i \leftarrow 1 \text{ to } N \text{ do}
22: \hspace{2em} \text{// Compute the final position}
23: \hspace{3em} (w, p'') \leftarrow \text{TANGENTSSTEP\((p', \Delta p', t)\)}
24: \hspace{2em} \text{// Compute the new direction}
25: \hspace{3em} (-, \Delta p'') \leftarrow \text{TANGENTSSTEP\((p', \Delta p', t)\)}
26: \hspace{2em} \Delta p' \leftarrow \frac{\|\Delta p''\|}{\|\Delta p'' - p''\|}
27: \hspace{3em} p' \leftarrow p''
28: \hspace{2em} t \leftarrow w
29: \hspace{1em} \text{end for}
30: \hspace{1em} \text{return} \(t, p', \text{atan2} \left(\Delta p'\right)\)
31: \textbf{end procedure}
Figure 2: More samples for the patterns resulting from the interpolation of the simulation parameters regulating the agents’ movement speed and the pheromone’s evaporation rate. Refer to Figure 10 from the main article.
Figure 3: More samples for the patterns resulting from the interpolation of the simulation parameters regulating the sensor angle and distance.
3. Patterns for materials

Figure 11 in the main paper shows three examples of different patterns that can arise from the slime mold algorithm on surfaces by tuning the simulation parameters. The materials used for the examples are fairly complex, and they come from the mixing of different shaders. However, the visible patterns are the result of a slime mold evolution, and we show them in Figure 4. The top texture has been used as a filter, for mixing two shaders and produce lava rivers on a rock, and as a displacement map for enhancing the details. We did the same for the bottom texture when producing the moss material over the bark. The central texture has been used as an inverse bump map on a metallic material to produce scratches and engravings.

Tables 2, 3 and 4 from the main paper summarize quantitative results on the algorithm’s performance. The results are averaged over a sample of triangle meshes with various resolutions. We show the sample of 10 meshes used for these experiments in Figure 5 and we summarize their statistics in Table 1.
Figure 5: The set of meshes used for testing performance.

<table>
<thead>
<tr>
<th>mesh</th>
<th>num. verts</th>
<th>num. edges</th>
<th>num. tris</th>
</tr>
</thead>
<tbody>
<tr>
<td>bunny stripes</td>
<td>18k</td>
<td>55k</td>
<td>37k</td>
</tr>
<tr>
<td>bunny</td>
<td>35k</td>
<td>104k</td>
<td>69k</td>
</tr>
<tr>
<td>dragon</td>
<td>32k</td>
<td>95k</td>
<td>63k</td>
</tr>
<tr>
<td>gargoyle</td>
<td>20k</td>
<td>59k</td>
<td>39k</td>
</tr>
<tr>
<td>torus</td>
<td>2k</td>
<td>7k</td>
<td>5k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mesh</th>
<th>num. verts</th>
<th>num. edges</th>
<th>num. tris</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphere</td>
<td>32k</td>
<td>98k</td>
<td>65k</td>
</tr>
<tr>
<td>owl</td>
<td>39k</td>
<td>118k</td>
<td>79k</td>
</tr>
<tr>
<td>cat</td>
<td>64k</td>
<td>193k</td>
<td>129k</td>
</tr>
<tr>
<td>fertility</td>
<td>10k</td>
<td>30k</td>
<td>20k</td>
</tr>
<tr>
<td>roman bust</td>
<td>14k</td>
<td>42k</td>
<td>28k</td>
</tr>
</tbody>
</table>

Table 1: Statistics of the sample meshes shown in Figure 5.