Evolving Mazes from Images

Liang Wan[1,2], Xiaopei Liu[1], Tien-Tsin Wong[1], and Chi-Sing Leung[2]

[1]The Chinese University of Hong Kong [2]City University of Hong Kong

{xpliu, ttwong}@cse.cuhk.edu.hk {liangwan,eeleungc}@cityu.edu.hk

Abstract—We propose a novel *reaction diffusion (RD) simulator* to evolve *image-resembling* mazes. The evolved mazes faithfully preserve the salient interior structures in the source images. Since it is difficult to control the generation of desired patterns with traditional reaction diffusion, we develop our RD-simulator on a different computational platform, cellular neural networks. Based on the proposed simulator, we can generate the mazes that exhibit both regular and organic appearance, with uniform and/or spatially varying passage spacing. Our simulator also provides high controllability of maze appearance. Users can directly and intuitively "paint" to modify the appearance of mazes in a spatially varying manner via a set of brushes. In addition, the evolutionary nature of our method naturally generates maze without any obvious seam even though the input image is a composite of multiple sources. The final maze is obtained by determining a solution path that follows the user-specified guiding path. We validate our method by evolving several interesting mazes from different source images.

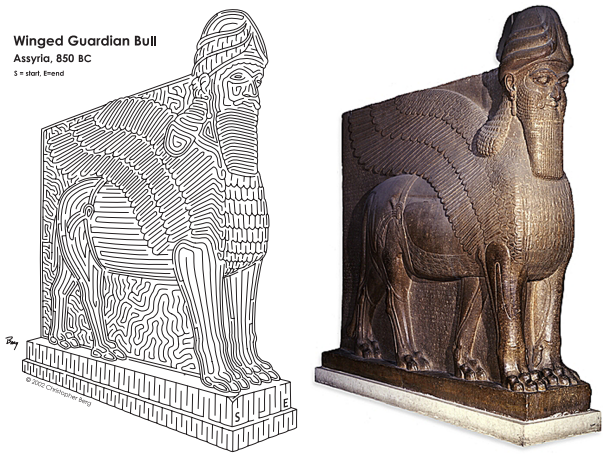
Index Terms—maze, multi-scale RD-simulator, cellular neural networks, intuitive user controls

1 Introduction

Creating mazes has a long history in human civilizations worldwide. It has been widely applied to different domains including, architectural decorations, design of botanical gardens [1], artistic expressions [2], or even puzzles in daily for entertainment [3][4]. As an impressive form of art, some mazes are intentionally designed to resemble images [5][6]. For example, the maze created by the artist (Figure 1(a)) depicts both the object silhouettes and salient interior structures in the source image (Figure 1(b)). Currently, most quality image-resembling mazes are manually designed due to the organic structures and complexities in the source image.

There have been several attempts to automate the creation of image-resembling mazes. By iteratively bending and extending the initial curves, Pedersen and Singh [7] simulated the organic labyrinth. To model more general mazes, Xu and Kaplan [8] assigned appropriate maze templates to the manually segmented regions. These methods can assist designers to create compelling mazes. However, they may not preserve the salient interior structures in the source image (especially when the structure is complex, such as the patterns in the bull wings in Figure 1(b)), and hence may harm the resemblance.

On the other hand, the classical mathematical model of reaction diffusion can create maze-like stripe patterns [9][10][11], on which we could rely to construct solvable mazes. Reaction diffusion, however, does not resemble source images and its multiple parameters are difficult to control. Motivated by the potential of reaction diffusion for maze generation, we propose a novel method to *evolve* image-resembling stripe patterns that



(a) manually designed maze (b) source image

Fig. 1. Example of an image-resembling maze: (a) the manually designed maze (image courtesy of Christopher Berg); and (b) the reference image.

preserve salient interior structures as well as the silhouettes in the source images (Figure 15(c) demonstrates our evolved maze from Figure 1(b)). The core of our algorithm is a novel RD-simulator [1] under the framework of cellular neural networks (CNN) [12]. It is capable of producing stripe patterns that range from organic to regular as well as those with uniform or spatially varying passage spacings.

With our RD-simulator, the generation of stripe patterns can be fully automatic or with optional interactive user control. Our method does not require tedious tuning of chemical parameters. Users can directly and intuitively "paint" (adjust) the passage spacing, the wall

*Corresponding author: Tien-Tsin Wong

1. Our specific RD-simulator in cellular neural networks is not a real reaction diffusion process, as its underlying dynamics is different from a real reaction diffusion. This is why we call it a "simulator."

and the passage, and the organic (random) appearance in a spatially varying manner via a set of brushes. The system then evolves the maze-like pattern to obtain the desired appearance. Finally, the system creates a solution path based on the user-specified guiding curve, making the stripe pattern actually a maze. As sophisticated methods for solution path construction have been developed [8], we are more concerned about the generation and intuitive control of image-resembling stripe patterns in this paper.

2 Related Work

Pedersen and Singh [7] proposed a geometric attraction-repulsion model that evolves a simple shape into an organic labyrinth, mimicking the work of artist Morales [2]. Traveling salesman art [13] is visually similar to this maze style. Xu and Kaplan [14] created one type of abstract mazes by arranging vortices. To model more general mazes, they developed a set of procedural algorithms [8], each producing a different maze template. By feeding the appropriate templates to the pre-segmented image regions, the source image can be resembled. However, to preserve the salient structures in the source image, their method requires the image to be carefully segmented by the designer. Such segmentation can be very tedious for images with complex interior structures like Figure 1(b). In contrast, our method can automatically preserve the salient structures during the evolution process.

Reaction diffusion has been developed to explain the formation of biological patterns [9][15][16], such as zebra stripes and seashell patterns. In this model, two or more chemicals diffuse and react with each other until an equilibrium is reached. Reaction diffusion can produce different patterns, which is verified by Witkin and Kass [11]. In particular, we are interested in labyrinthine patterns that can be created with Meinhardt model [10], Gray-Scott model [17] and Fits Hugh Nagumo system [18]. These models require careful parameter tuning which is difficult for ordinary users. The work of [19] provides more convenient interfaces for pattern control, but it may involve intensive user interventions. What is more, they all do not resemble any source image. In this paper, we propose a CNN-based RD-simulator. In addition to avoiding the tedious parameter-tuning, our RD-simulator offers sufficient control to resemble the source image, as well as to manipulate the maze properties in a spatially varying manner.

3 RD-simulator

A maze is a set of connected passages delimited by walls. In this sense, the stripe pattern generated by reaction diffusion is very similar to a maze, except that the passages may not be connected and solvable. Motivated by this observation, we first feed the source image to our RD-simulator to *evolve* an image-resembling *stripe pattern*. With the user-specified guiding curve, our system then constructs a solution path in the stripe pattern and yields a solvable *maze*. Figure 2 demonstrates the basic process of maze evolution. In our work, we are more focused on generating image-resembling stripe patterns and providing intuitive user control of maze appearance. As shown later, the two objectives can be effectively achieved by using our RD-simulator. Since the simulator is developed under the CNN framework [20], we first briefly introduce the basic idea of the CNN below.

3.1 Cellular Neural Networks

Originated in the field of integrated circuits, CNN is an attractive parallel computing paradigm similar to neural networks, with the difference that interaction is allowed only within a finite local neighborhood. Since its invention in 1988, CNN has evolved into various structures to cover a broad class of problems, including image processing, solving partial differential equations, modeling biological vision, etc. Among the common CNN structures, we adopt the autonomous CNN which is capable of generating various patterns [12].

A typical autonomous CNN is a 2-dimensional regular grid of identical computational cells that are locally connected with the neighboring cells (each cell represents an image pixel in our application). Suppose the state of one cell is X, and its response is Y. The CNN dynamics is a nonlinear equation of X and Y of each cell within the neighborhood η, as given by

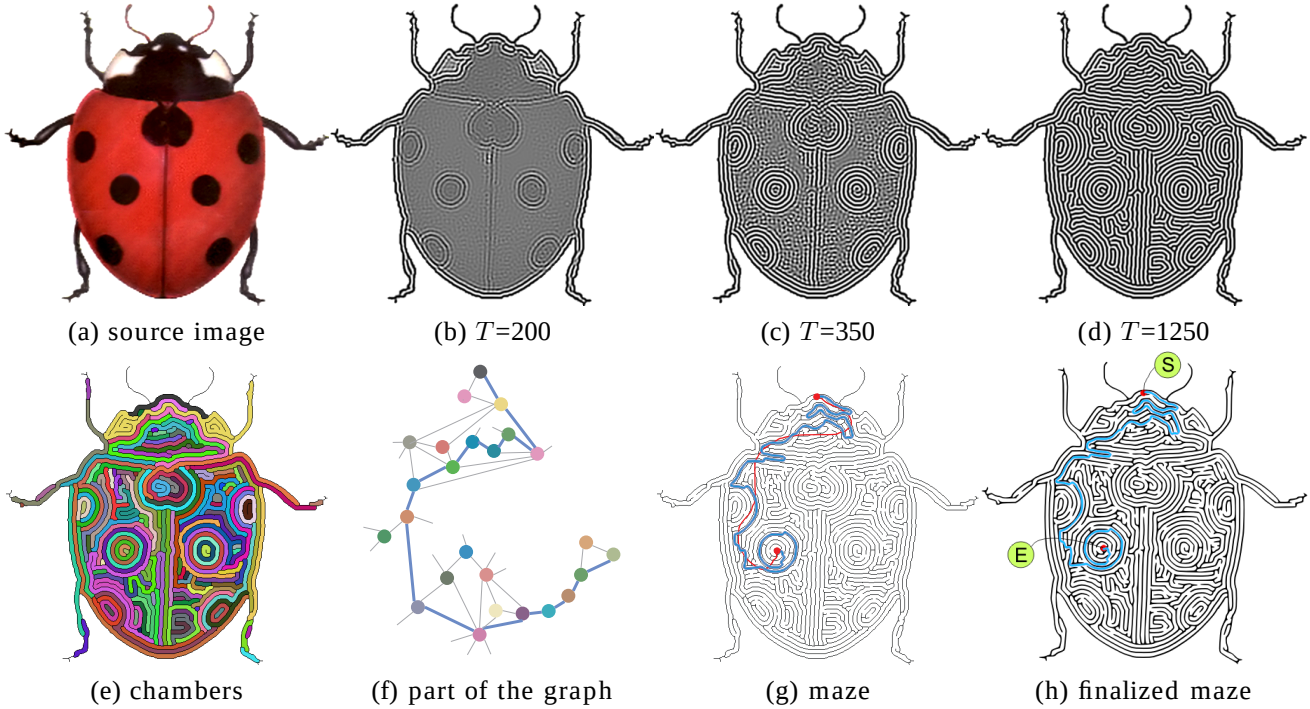
$$\frac{dX_{i,j}^{(t)}}{dt} = -X_{i,j}^{(t)} + \sum_{k,l\in\eta(i,j)} a_{k-i,l-j} Y_{k,l}^{(t)} + I_{i,j}, \qquad (1)$$

where (i,j) defines a grid point associated with a cell on the grid; I is the input bias; t is the time parameter; $\mathbf{A} = \{a_{k,l}\}$ is the cloning template which specifies the interacting dynamics, and it is usually spatially invariant for all cells. The output Y is determined from the state X via a non-linear transfer equation, which is usually set as:

$$Y_{i,j}^{(t)} = \frac{1}{2}\left[|X_{i,j}^{(t)}+1| - |X_{i,j}^{(t)}-1|\right], \qquad (2)$$

where $|\cdot|$ returns absolute value. Let $X^{(0)}$ be the initial state of X. The inputs $X^{(0)}$, I and the output Y for each cell are represented by values in the range $[-1,1]$ (e.g., +1 is white, -1 is black, and gray-scale values are in between).

It has been found that with appropriate cloning templates, the autonomous CNN can eventually reach a stable equilibrium state and the output Y forms some kinds of patterns. However, most existing works usually assume that $X^{(0)}$ should be a small noise and $I = 0$, and hence create a *random* pattern only. It is also noticed that an arbitrary cloning template may lead to chaos instead of a stable pattern. In fact, most pattern-generating cloning templates are discovered by the theoretical study of the dynamic evolution of the autonomous CNN.



(a) source image (b) T=200 (c) T=350 (d) T=1250

(e) chambers (f) part of the graph (g) maze (h) finalized maze

Fig. 2. Maze evolution. Top row: (a) is the source image. The rest are the intermediate results during our reaction-diffusion simulation after T iterations: (b) T=200, (c) T=350, and (d) T=1250. The bottom row shows how the maze in (g) is constructed from the stripe pattern in (d), given the red guiding curve. (e) is the color-coded chamber image. (f) is part of the associated graph of (e) showing the solution path from the starting ('S') to the end ('E') points. The solution path is automatically selected by our system and is colored in blue in both (f) and (g). (h) is the final vectorized maze image. Note that the salient interior structures, such as the spots and the dark separation line on the ladybug, are faithfully retained in the maze image.

3.2 Basic RD-Simulator

To generate a stripe pattern that resembles a source image, we employ the following cloning template [12],

$$\mathbf{A} = \begin{bmatrix} -0.25 & -1.0 & -1.5 & -1.0 & -0.25 \\ -1.0 & 2.5 & 7.0 & 2.5 & -1.0 \\ -1.5 & 7.0 & -23.5 & 7.0 & -1.5 \\ -1.0 & 2.5 & 7.0 & 2.5 & -1.0 \\ -0.25 & -1.0 & -1.5 & -1.0 & -0.25 \end{bmatrix}, \quad (3)$$

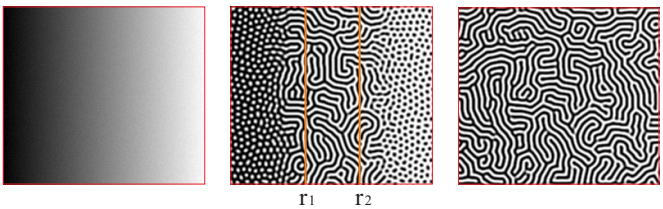
which defines a neighborhood η of size 5×5. Like a traditional reaction-diffusion model, this cloning template provides both local activation (as the nearest neighbor interactions are positive) and long range inhibition (as the following neighbor interactions are negative). Hence, during the evolution, the state X can finally form a reaction-diffusion like pattern. Note that the original approach [12] only produces a random stripe pattern. In order to resemble a source image P, we carefully utilize the autonomous CNN in such a way that P is persistently fed into the system by the setting,

$$X^{(0)} = c_1 P_m, \quad I = c_2 P_m, \quad (4)$$

where $P_m = \text{gray}(P)$ is the grayscale version of image P with pixel values linearly remapped to gray levels in $[r_1, r_2]$.

The first setting $X^{(0)} = c_1 P_m$ helps us to maintain the salient structures in the source image. The second setting $I = c_2 P_m$ is to reinforce the reaction and diffusion around salient edges. The linear remapping is specified because when feeding the system with P, the image regions with pixel values around 0.5 can evolve to be stripe patterns while the regions with pixel values close to 0 or 1 evolve to be dot patterns (as demonstrated in Figure 3(b)). In particular, we set the parameters $r_1 = 0.396$ and $r_2 = 0.588$ to enforce the generation of stripe patterns in the whole image. We also choose the coefficients $c_1 = 2$ and $c_2 = 1$ empirically, with which the resulting stripe pattern represents the source image visually well. To apply the CNN, P_m has to undergo the mapping $f(x) = 2x-1$ before the evolution. Accordingly, the output Y within the range $[-1, 1]$ requires the inverse mapping $f^{-1}(x) = (x+1)/2$.

The set of Equations (1)-(4) defines our basic RD-simulator. To help understand how it works, Fig-



(a) (b) (c)

Fig. 3. The remapping setting. (a) The input image P is a gray ramp contaminated by uniform noise. (b) Without remapping, the gray values below r_1 result in white dot patterns while those above r_2 lead to black dot patterns. (c) With remapping, an entire stripe pattern results.

ures 2(b)-(d) visualize the evolution process given the source image in Figure 2(a). It is obvious that the salient structures like spots on the ladybug enforce stronger impact on the generation of stripe patterns. As a result, the salient structures are automatically and faithfully retained when an equilibrium is reached in Figure 2(d).

Unlike previous approaches using separate procedures to generate regular and organic mazes, our basic RD-simulator can create both of them within the same process. When the source image contains mainly the smooth regions without much noise, the evolved stripe pattern is rather regular and its appearance is determined by the salient interior edges and region silhouettes (as demonstrated in Figure 4(b)). It is similar to those using level-set method [21] or concentric contour approach [8]. On the other hand, when the source image contains rough interior regions, the rough contents will compete with the salient edges or silhouettes during the evolution, and will finally lead to an organic stripe pattern. For example, Figure 4(c) shows the results by simply introducing random noise to the source images (perturbing the grayscale level in the interior regions).

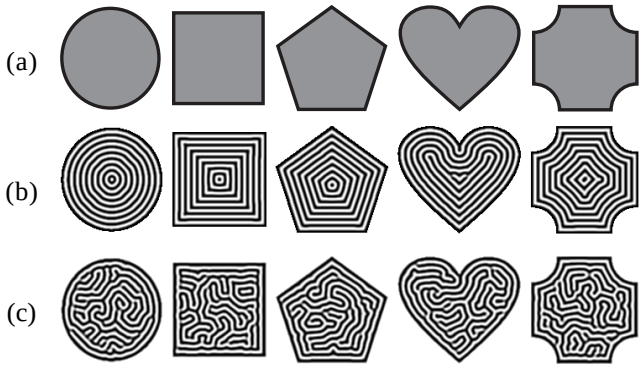


Fig. 4. Regular and organic stripe patterns: (a) the source images; (b) the corresponding resulting regular stripe patterns; and (c) the evolved stripe patterns after adding noise to the source images in (a).

3.3 Multi-scale RD-simulator

The basic RD-simulator, however, generates stripe patterns with a fixed passage spacing, which is independent of image size. This is because the 5×5 neighborhood is used for each pixel. Here we define the passage spacing as the distance between the centers of black walls. It is known that the maze designers usually convey the sense of perspective effect by using larger passage spacings in the near and smaller passage spacings in the distant. They may also adjust the passage spacing to represent different structure complexity, or to reproduce the spatially varying image tone.

To change the passage spacing, one may suggest employing other stripe-generating cloning templates with different sizes. However, it is quite difficult to theoretically deduce the cloning templates of larger sizes. Moreover, this scheme requires the template size to be a discrete integer, making it incapable of obtaining arbitrary passage spacing. A possible solution is to use the 5×5 cloning template and scale the source image. But such solution can hardly support spatially varying spacing. To address this problem, we propose a *multi-scale* RD-simulator. Its basic idea is to adjust the size of the pixel neighborhood during the convolution while using the current 5×5 cloning template, as illustrated in Figure 5. This is equivalent to scaling the image and hence can change the passage spacing. By controlling the neighborhood size locally, it is also possible to seamlessly generate different passage spacings at different regions.

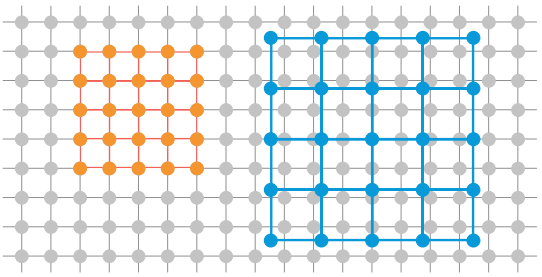


Fig. 5. The multi-scale RD-simulator generates spatially varying passage spacing by spatially adjusting the size of the neighborhood for convolution. The convolution neighborhood in orange has the minimal size of 5×5, while the blue one is modulated by a larger S value.

The multi-scale RD-simulator is described by the following equation,

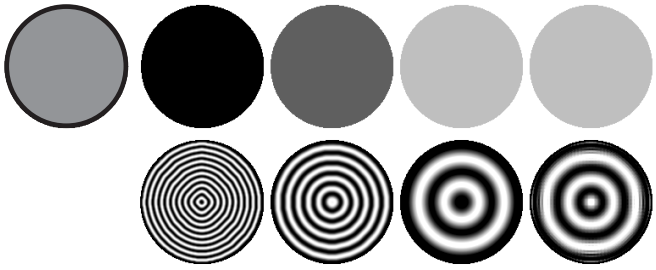
$$\frac{dX_{i,j}^{(t)}}{dt} = -X_{i,j}^{(t)} + \sum_{k,l \in \eta(i,j)} a_{k-i,l-j} Y_{\tilde{k},\tilde{l}}^{(t)} + I_{i,j}. \quad (5)$$

The grid point $(\tilde{k}, \tilde{l})$ in the neighborhood of (i, j) is given by,

$$\tilde{k} = i + \frac{k-i}{1-S_{i,j}}, \quad (6)$$

$$\tilde{l} = j + \frac{l-j}{1-S_{i,j}}, \quad (7)$$

where the parameter $S_{i,j} \in [0, 1)$ is a scale value at pixel (i, j). Intuitively, larger $S_{i,j}$ values lead to larger passage spacings. When $S_{i,j} = 0$, the multi-scale RD-simulator reduces to the basic RD-simulator (Equation 1). When $S_{i,j} > 0$, Equations 6 and 7 define a convolution neighborhood larger than the size 5×5, in which the neighboring point values can be computed by interpolation. Note that the multi-scale RD-simulator has different evolution dynamics from the basic simulator. For nearby image pixels, the set of points in their convolution neighborhoods may not overlap when $S_{i,j} > 0$. However, we can conceptually split the multi-scale simulator into multiple basic simulators, each corresponding to a part of the image. Their convolution neighborhoods are regularly overlapped. By coupling these basic simulators through the interpolation computation, the multi-scale RD-simulator can finally converge to a stable equilibrium. Figure 6 demonstrates how the passage spacing changes with respect to the scale.



(a) input (b) S=0 (c) S=0.373 (d) S=0.75 (e) S=0.75

Fig. 6. Varying passage spacing. Top row: (a) is the source image while (b)-(e) are the scale maps with constant values S=0, 0.373, 0.75 and 0.75. Bottom row: the evolved stripe patterns exhibit increasing passage spacings. (e) shows aliasing artifacts due to sampling the neighborhood for convolution. (b)-(d) are free from such artifacts by applying Gaussian filtering to the state X in each iteration.

The multi-scale RD-simulator, however, may cause aliasing artifacts due to the sampling of 5×5 neighborhood during the convolution (as shown in Figure 6(e)). Larger $S_{i,j}$ values may give more severe artifacts. This is because large scale values bring weak coupling in the evolution dynamics. To strengthen the coupling, we apply a spatially varying Gaussian filter to the state X of each cell in each iteration, with a larger Gaussian kernel for a larger S value. This is controlled by setting the Gaussian filter parameter $\sigma = \alpha - \beta \cdot (1 - S_{i,j})$. In all our experiments, we found that $\alpha = 0.6$ and $\beta = 0.6$ are sufficient to avoid aliasing. The results in Figures 6(b)-(d) are obtained by applying this Gaussian filtering approach in each iteration.

With the multi-scale RD-simulator, users can modulate the passage spacing in a spatially varying manner by providing or drawing a scale map, similar to that employed in [7]. In fact, we can estimate the passage spacing $D_{i,j}$ analytically. It is determined by the scale value $S_{i,j}$ where $D_{i,j} = D^0_{i,j}/(1 - S_{i,j}) - 1$ (this relation holds when the Gaussian filtering is applied). Here, we observe that $D^0_{i,j} = 8$ when a convolution neighborhood of size 5×5 is used. Since the passage spacing tends to infinity when $S_{i,j}$ is close to 1, we limit the value of $S_{i,j}$ in the range of $[0, 0.9]$, in practice. In addition, $S_{i,j} = 1$ is used as a flag to mask out the unwanted image region.

Figure 9(f) shows a result. In this example, the seahorse body has a larger structure than its head and tail, so we adjust the passage spacing accordingly (see the scale map in Figure 9(b)). As shown, the continuous change in the scale value yields a smooth change in the passage spacing, e.g., the transition between the body and head / tail parts. Even with a sharp transition in scale values, the abrupt change in the structure between the fin and body is also maintained in the evolved stripe pattern. We further demonstrate the adjustment of scale values to create the perspective effect on the ground in the "pyramid" example (Figure 14). Examples of reproducing image tone with spatially varying passage spacing can be found in the middle column of Figure 11.

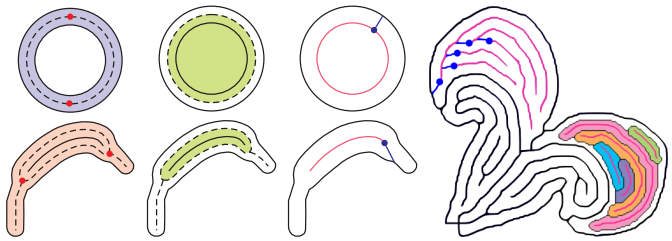
4 Creating Mazes from Stripe Patterns

The evolved stripe pattern may not be a solvable maze since it may contain many isolated passages. To convert it to a maze, we construct a planar graph from the stripe pattern and build a solution path on the graph.

4.1 Graph Construction

To construct the planar graph, we first binarize the stripe pattern and skeletonize the black walls by tracing the central lines [22]. Next we detect each white passage (chamber) and represent it as a node in the graph. Adjacent chambers separated by a black wall are connected with an edge in the graph. Figures 2(e) and (f) show the color-coded chambers and part of the associated graph respectively.

Some chambers, however, may have loops inside, bringing circular paths in local regions. As shown in Figure 7(a), when the chamber centerlines form closed curves, there exist two possible paths between arbitrary two points (in red color) on the chamber centerlines. To remove the loops, we need to detect which chambers contain loops and then break the loops by adding auxiliary walls.



(a) (b) (c) (d)

Fig. 7. Loop removal. (a) illustrates two common cases in which the color-coded chambers contain loops inside. (b) shows that the chamber centerlines (in dashed curves) form closed regions. In (c), we select appropriate points and add auxiliary walls. Note the loops are successfully removed. (d) demonstrates a more complicated example.

For loop detection, instead of checking whether the chamber centerlines form the closed curves, we detect whether they enclose isolated regions (or loop regions), as shown in green color in Figure 7(b). Here, we exclude the chamber walls in the detection. Then, if the chamber wall in the loop region (the inner wall) is a closed curve (such as the upper example in Figure 7(c)), we randomly select a point on the wall and connect it to its nearest point on the chamber wall outside the loop region (the outer wall). If the inner wall is an open curve (such as the lower example in Figure 7(c)), we choose one ending point on the wall and elongate it along the curve direction until it reaches the outer wall. Figure 7(d) shows a more complicated example where many loops exist in one chamber (see the color-coded loop regions). To ensure all the loops to be removed, we require that the auxiliary walls must reach the outer walls. For this

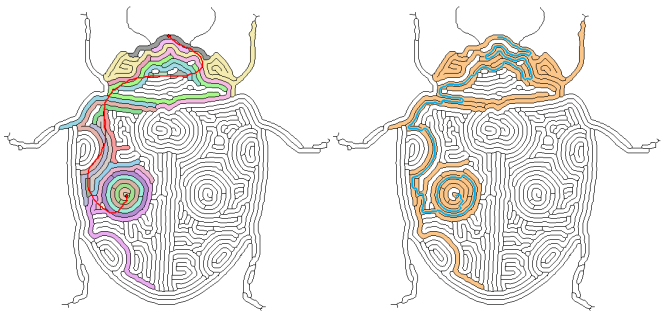
purpose, we iteratively detect the loops and convert the inner walls with auxiliary walls added to the outer walls in each iteration.

Note that the process to remove the loops takes place in chambers. It does not introduce more chambers or change the layout of the chambers. Hence the planar graph remains the same.

4.2 Building Solution Path

Given the planar graph, constructing a solvable maze can be achieved by traversing the graph, and building a solution path is equivalent to finding a path in the graph. Here, we consider the perfect maze, a common type in maze production, which contains no circular paths. Then maze construction can be easily done by establishing a random spanning tree of the graph using any standard maze algorithm. To offer the user an intuitive control over the routing of the solution path, we allow the sketching of a guiding curve on the top of the stripe pattern. We then try to build a solution path that follows but may not exactly match the guiding curve.

As shown in Figure 8(a), the user-drawn guiding curve suggests which chambers are on the solution path. However, the curve may visit one chamber multiple times, as the chamber could be long and complex in the layout. Consequently, cycles may occur in the solution path. To remove such unexpected cycles, we require one chamber be visited only once. By omitting the nodes (chambers) between the redundant chambers, we yield a solution path without cycles in the graph (see Figure 8(b)). Then the entire spanning tree of the graph is constructed starting from the obtained solution path. To get the solvable maze, we link two adjacent chambers on the tree by breaking the wall (edge) that separates them at an arbitrary point. Figures 2(g) and (f) show the solution path in the maze and in the corresponding graph respectively. Another example showing the guiding curve and the solution path can be found in Figure 14.



(a) (b)

Fig. 8. Solution path construction. (a) shows the user-drawn guiding curve (in red color). It determines which chambers may be on the solution path. (b) shows the blue solution path without cycles as well as the chambers (in orange) contributed to the solution path. Note the final solution path follows but not exactly match the guiding curve.

Careful readers may notice that the guiding curve was also adopted by Pedersen and Singh [7]. Their work differs from ours in that they used the guiding curve to guide the dynamic evolution of the labyrinth, while we use it after the dynamics of obtaining the stripe pattern. Xu and Kaplan [8] allowed users to sketch a solution tree with which they needed an image segmentation to generate the final solution path.

4.3 Vectorization

So far, all the computations are accomplished in the domain of raster image. To avoid jaggy walls due to aliasing, we vectorize the constructed maze [23]. Then the vectorized maze (Figure 2(h)) can be scaled arbitrarily without unpleasant jaggy walls.

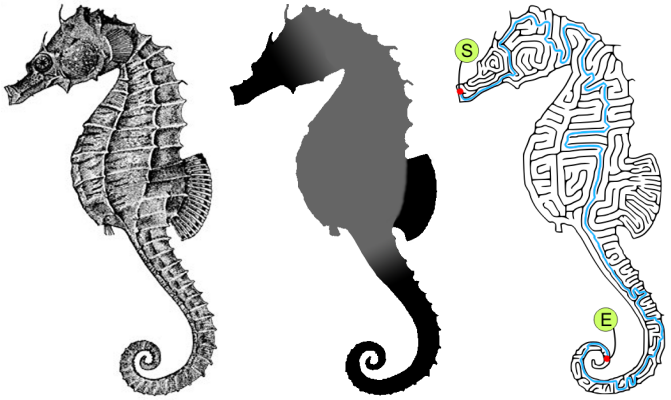
5 User Control

Our RD-simulator provides a convenient platform to control the appearance of stripe pattern. By adjusting the input parameters in appropriate ways, the user can intuitively, directly and locally modify the pattern appearance. In our system, we provide a paint-brush interface to facilitate the user intervention. Here we highlight its major functionalities.

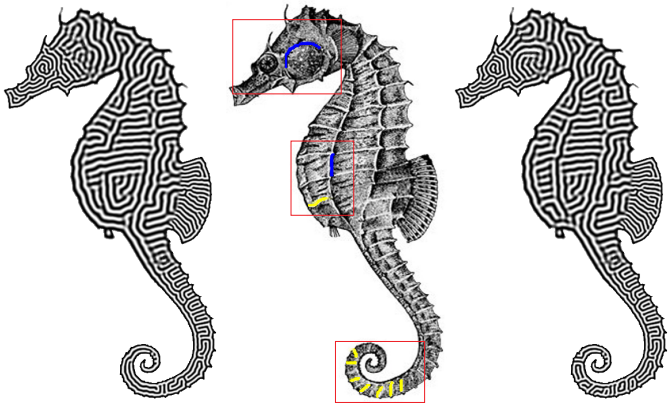
5.1 Local Structure Manipulation

To preserve the desired structure in the source image, we provide a *wall brush* and a *passage brush*. For instance, in Figure 9(a), although some edges are not sufficiently strong to guarantee the generation of walls in the stripe pattern (see Figure 9(d)), they may be perceptually important. To preserve them, users can directly draw walls with the wall brush (the blue stroke) as illustrated in Figure 9(e). After re-running the RD-simulator, the additional walls seamlessly merge with the surrounding stripes (Figure 9(f)). Similarly, users can explicitly draw passages via a *passage brush* (the yellow strokes). The wall and passage brushes are realized by setting the *state* value X of each pixel on the updating strokes as -1 and $+1$ respectively in each iteration (recall that -1 corresponds to black and $+1$ corresponds to white). In practice, one-pixel thickness is sufficient for the updating strokes. The brush strokes in Figure 9(e) are thickened for the illustration purpose.

Besides, users can provide additional patterns, such as a brick pattern or dot pattern, to replace a region in the source image with a *pattern brush*. This generates stripe patterns with the user-defined structure naturally *fused* with the intrinsic structures in the source image. Figure 14 demonstrates such fusion of a real pyramid photo with regular brick patterns. In this example, the stone bricks in the original photo is too small to generate meaningful brick structures in the final maze. The pattern-based scheme is especially useful for design purposes. Users can create a variety of maze types by replacing image regions with their favorite patterns.



(a) source image (b) scale map (c) final maze from (f)



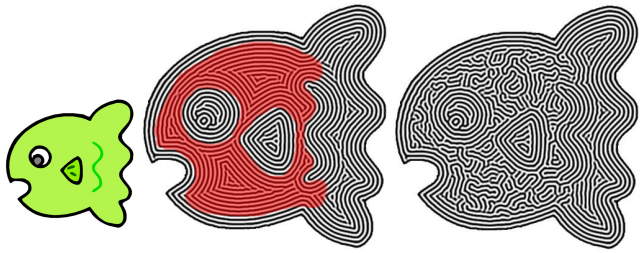
(d) stripe pattern from (a) (e) structure enhanced (f) stripe pattern from (e)

Fig. 9. Local structure enhancement: (a) the source image; (b) the scale map; (c) the final maze created from (f); (d) the stripe pattern directly evolved from (a) may not capture some edges in the source image; (e) the original image is enhanced. (f) the stripe pattern evolved from (e) better resembles the source image.

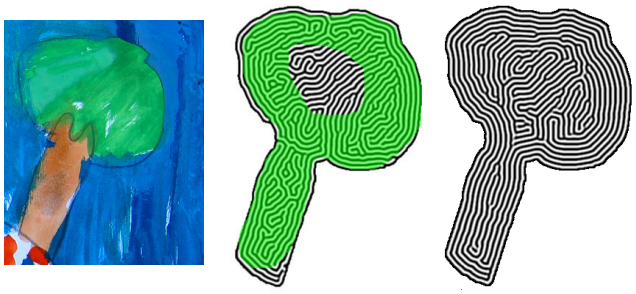
Note that similar to the case of natural images, only salient structures in the patterns can be retained via the RD-simulator. From our experience, simple and black-and-white patterns are preferable. One more example can be found in Figure 15. A cross pattern is applied on the chest of the "Winged Guardian Bull".

5.2 Regular/Organic Control

As mentioned before, noise can be introduced to make the maze more organic. Therefore we allow the user to enforce a smooth region to evolve into organic stripe pattern using an *organic brush* (the red stroke). This is simply realized by adding a strong white noise to the brushed region in the source image. Similarly, users can also use a *regular brush* (the green stroke) to smoothen the noisy region so as to generate more regular stripe patterns. This smoothing is achieved by applying the Gaussian filtering to the brushed region. Figure 10 shows the organic and regular brushes in action.



(a) organic brush



(b) regular brush

Fig. 10. Organic and regular brushes. The top and the bottom rows show the organic and regular brushes in action. The left, middle, and right columns show the source images, the initial stripe patterns with user brushes, and the updated stripe patterns, respectively.

5.3 Passage Spacing Manipulation

To generate a stripe pattern with the desired local passage spacing, we offer the *scale brush*. Once a desired scale value (grayscale level) is chosen, users can fill a region in the scale map with the selected value. In this way, users can intuitively and directly edit the scale map in a spatially varying manner. To avoid unnatural seams between the regions with different passage spacings, a smooth transition of scale values is needed in the scale map. This can be achieved by blurring the scale map. Examples using manually designed scale maps can be found in Figures 9, 14 and 15. Note that the evolved mazes exhibit spatially varying passage spacings in a natural way.

Besides the hand-drawing scale brush, we also developed two methods to automatically generate the scale map. One is based on the image tone and the other is based on the structure complexity. For both methods, we first segment the source image [24]. In the tone-based method, we compute the average intensity P' for each segment. The scale map is then calculated by a linear mapping, $S = \tau \cdot P'$, where $\tau = 0.6$. We also excessively blur the scale map with a Gaussian filter to avoid abrupt change in the scale map. The middle column of Figure 11 shows a scale map (top) created with this method and its corresponding maze (bottom). Note the passage spacing decreases in the region with lower image tone, which helps to reproduce the image tone.

Similarly, we can generate the scale map according to the structure complexity. Here we want to introduce smaller passage spacing for regions with higher structure

Fig. 11. Automatic generation of scale maps. Left column: the source image (top) and its segmentation map (bottom). Middle column: the scale map (top) generated based on intensity and the evolved maze (bottom). Right column: the scale map (top) generated based on the structure complexity and the corresponding maze (bottom).

complexities. It is observed that complex regions usually correspond to the smaller segments. Hence, we can simply assign the segment size as the scale value of pixels within the segment and then map it to the range $[0, \tau]$. Again, Gaussian filtering is applied to avoid the abrupt change. The right column of Figure 11 shows a scale map (top) generated by this method and its corresponding maze (bottom). In the result, the passage spacing decreases when the structure complexity increases.

5.4 Wall Thickness Adjustment

Besides the passage spacing, we may change the wall thickness to reproduce the image tone. Although the walls in the stripe pattern have a width larger than 1, their thickness does not reflect the image tone (as shown in Figure 2(d)). So we simply discard the original wall thickness by thinning the walls to one-pixel width and take the resulting pattern as the base for wall thickness adjustment. Users can optionally turn on this thickness control.

Let the desired wall thickness be W. The relationship between wall thickness W, passage spacing D, and the pixel intensity $\tilde{P}$ can be modeled as $W = D(1-\tilde{P})$ [8]. As mentioned before, the passage spacing D is determined by $D = D^0/(1-S) - 1$. We also impose a minimal wall thickness ζ and a minimal passage thickness ξ to avoid the passage from being stuck. Hence, the valid range of wall thickness W is $[\zeta, D-\xi]$. Given the centerlines of the maze walls, we can render a maze with spatially varying wall thickness by using a brush with a varying size W. Figure 12(c) shows the rendered maze which resembles the tone in the source image in 12(a).

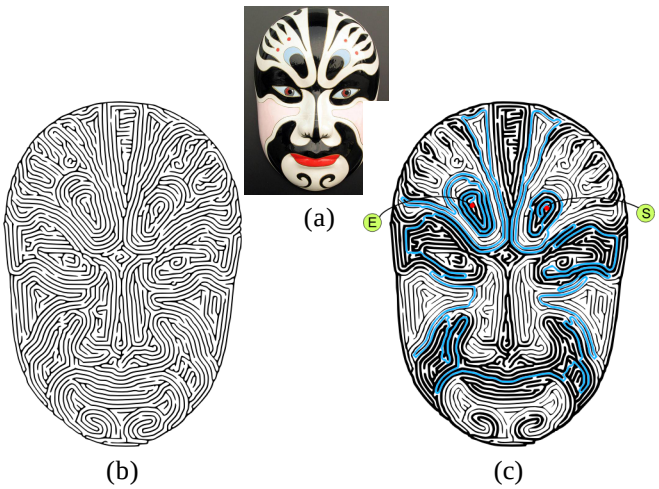


Fig. 12. Maze with varying wall thickness: (a) the source image; (b) the maze without spatially varying thickness and (c) the maze with spatially varying thickness that resembles the image tone.

6 Results and Discussions

In this section, we first validate our method using more examples, then we present the timing performance of our system and finally discuss the limitations of our method.

6.1 Experimental Results

Figures 2, 9, 11, 12, 14-17 demonstrate the results of the evolved mazes from different images. Note how the salient interior structures in the source images are well preserved in the mazes. Figure 15(c) shows a maze created from a picture depicting the sculpture of the "Winged Guardian Bull". The original pattern on the wings of the bull is convincingly preserved. Such an effect is actually hard to achieve using existing maze generation algorithms. In Figure 16, we blend a real-life leaf photo with the corresponding maze. Note that the maze exhibits clear structures of the main veins in the leaf photo. Figure 17 superimposes a maze of "Van Gogh Self Portrait" with the source image. Obviously, the evolved maze keeps the rich directional brush patterns in the image.

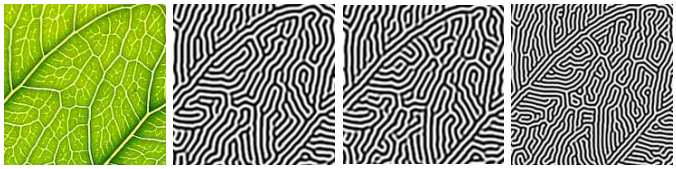
From our experience, our system is able to provide the users a flexible control over the maze appearance. For example, Figure 15(a) shows the regular brush applied to the legs and the organic brush applied to the background regions. We also modulate the scale map in order to represent different image structures, tones, or the perspective effect. In the pyramid maze (Figure 14), we edit the scale map at the bottom of the image, so that the perspective effect can be formed on the ground. The similar scale manipulation (Figure 15(b)) is applied to the base of the bottom stone in Figure 15(a). In addition, varying passage spacing can help to differentiate the objects in the image. In Figure 15(c), the bull body, legs and the background are clearly distinguished by assigning different passage spacings. Figure 14 demonstrates the fusion of multiple sources. Here, several brick patterns are laid over the real-world photograph. In Figure 15(c), by replacing a cross pattern at the front bull, an interesting pattern can be formed that naturally fuses with the surrounding. The drawing of several parallel walls on the beard of the bull also introduces pleasing maze patterns.

6.2 Timing Performance

Since the proposed RD-simulator (Equation 5) is highly parallelizable, we implement it with a GPU shader. Currently we build our system on a machine installed with Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz and nVidia GeForce 8800 GTX GPU. The main cost of generating a maze is on the stripe pattern evolution, which is an iterative process. To ensure the RD-simulator to reach a stable equilibrium, we empirically adopt the setting, 0.01 time increment and 3000 iterations, in all our experiments. The timing of the evolution relies also on the image size. For example, to evolve the maze in Figure 14, it takes about 35.7 seconds to obtain the stripe pattern for a valid image region of 475,736 pixels. For a mid-size image, like the seahorse in Figure 9, which has the size of 400×699 and a valid image region of 95,643 pixels, it only takes 11.6 seconds. On the other hand, the timing of converting a stripe pattern to maze depends on the image size (for loop removal) and the size of the planar graph associated to the stripe pattern (for building the solution path). The whole conversion normally takes less than 20 seconds given the user-specified guiding curve.

6.3 Discussion and Limitation

Our RD-simulator favors salient structures in the source image. This is reflected in two aspects. First, strong gradients enforce strong correlation between the image and the stripe pattern. In other words, the boundaries with strong gradients can defeat those with weak gradients during the dynamic evolution and dominate the resulting stripe pattern. Second, since the minimal passage spacing is $D^0 = 8$, small-scale image structure or texture pattern smaller than that cannot be preserved. For example, the bull in Figure 15(c) does not faithfully capture the face. Also, for the leaf in Figure 16, the evolved stripe pattern does not have a clear structure of those weak and fine leaf veins (the blowup shown in Figure 13(b)). By sharpening the image edges, the weak but large-scale image structures can become more apparent (Figure 13(c)), but the fine veins may still not be recognizable. To keep small-scale image structures, denser stripes have to be created, which can be achieved by up-sampling the source image (Figure 13(d)). This, however, may be inconvenient when users want to increase the stripe density only in local regions.

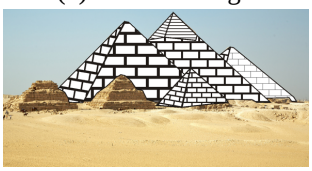


(a) (b) (c) (d)

Fig. 13. Structure preserved in stripe pattern. (a) shows one region of the leaf photo in Figure 16. (b) is the stripe pattern corresponding to (a). Note that both weak and fine leaf veins are poorly preserved. (c) By enhancing the edges in the source image, some weak vein structures become more apparent. (d) After scaling the image by 1.5, the stripe pattern provides a more recognizable structure even for the fine veins.

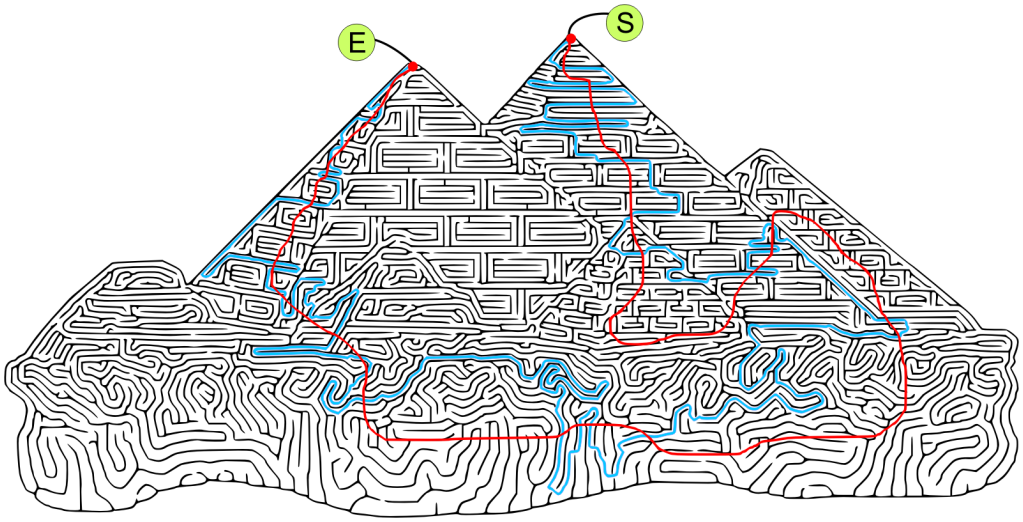
Another property of our method is that the RD simulation always grows concentric stripes on both sides of a strong feature unless there are other salient structures nearby. Although such propagation can be constrained by using a segmentation map, there is no intuitive way to avoid it. In addition, the RD-simulator only considers 2D information in nature and cannot handle complex 3D effects, even though we can adjust the scale map to fake the perspective effect to some extent. One may also

(a) source image



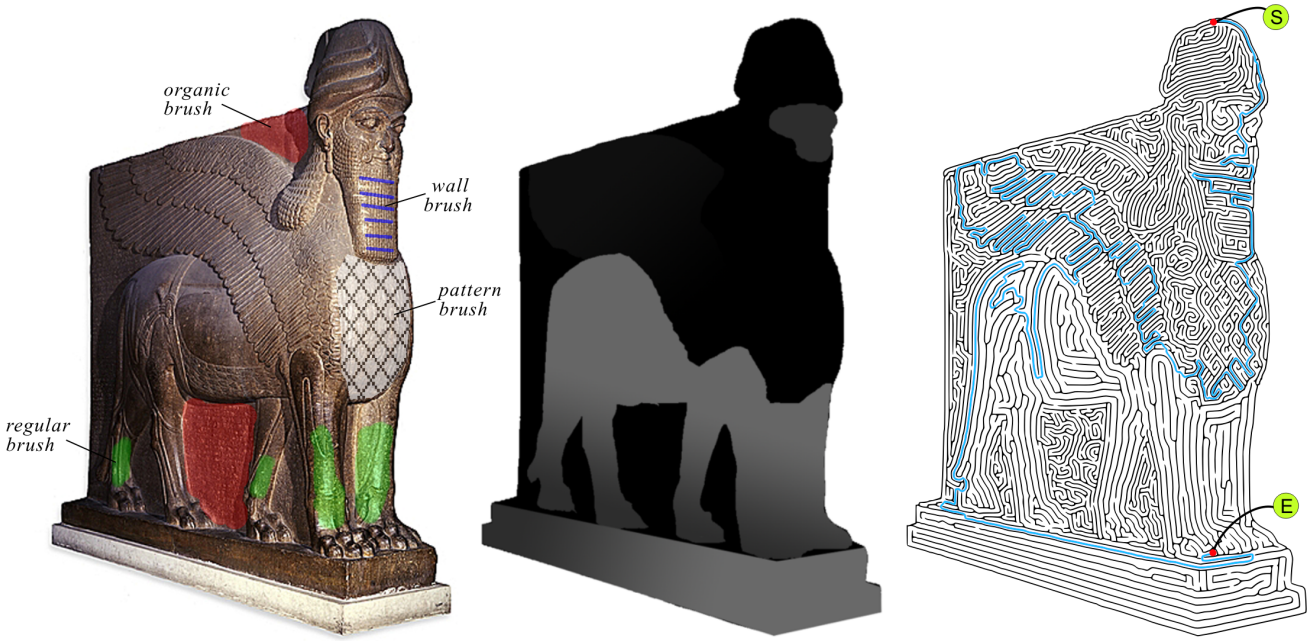
(b) edited source image

(c) scale map



(d) our evolved maze

Fig. 14. The maze of pyramid. (a) is the original image. (b) is the edited source image. (c) is the scale map to control the passage spacing. The evolved maze is shown in (d), with the red guiding curve and the blue solution path. Note that the maze preserves the salient interior structures of the pyramid image, while the transition from the edited parts to the original source image is natural in the final maze.



(a) source image with user control (b) edited scale map (c) our evolved maze

Fig. 15. The maze of "Winged Guardian Bull": (a) the source image edited with different brushes; (b) the scale map; (c) our evolved maze. Note how the salient structure on the bull wings is faithfully preserved in the result.

be curious whether our approach can faithfully generate a maze pattern with sharp right angled wall. This is dependent on the structures in the image. For example, the brick patterns on the pyramid in Figure 14(b) can evolve to right angled walls.

7 Conclusion

In this paper, we present a novel RD-simulator to evolve image-resembling mazes from the source images. Unlike traditional reaction diffusion models, we develop the (multi-scale) RD-simulator on the CNN computational platform. The evolved mazes can faithfully retain the salient interior structure in the images. In addition, our RD-simulator enables the direct and intuitive con-

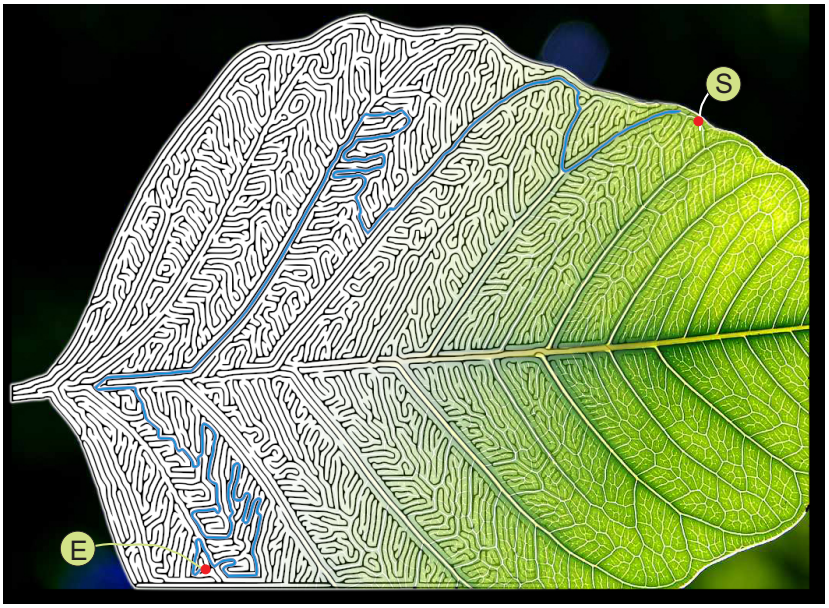


Fig. 16. A real leaf photo blended with the corresponding maze.

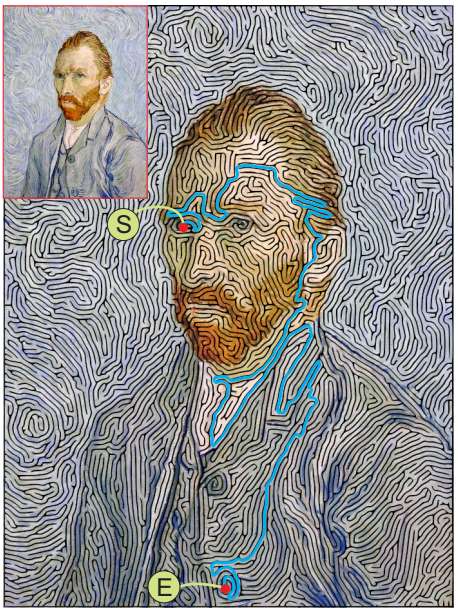


Fig. 17. The maze of "Van Gogh's Self Portrait" superimposed on the source image.

trol over the maze appearance. With a set of different "brushes", users can locally manipulate the maze structure, control the organic and regular look, and adjust the passage spacing.

Our method can be extended in several ways. First, we may speed up the stripe pattern refinement by locally applying the RD-simulator since the manipulation is usually performed in local regions. But we must be careful since local changes may eventually propagate to the other remote image regions. Second, we may break the wall near the intersection points between the sketched guiding curve and the chambers, other than at an arbitrary point which is what we have done currently. Third, the evolved stripe pattern may contain long chambers, which may be undesirable for some users. To solve this problem, we may construct grids on the stripe pattern before building the solution path. Readers can find more information about the work at

`http://www.cse.cuhk.edu.hk/~ttwong/papers/maze/maze.html`

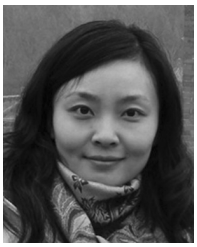
Acknowledgments

This project is supported by the Research Grants Council of the Hong Kong Special Administrative Region, under General Research Fund (Project No. CUHK 417107) and the Strategic Research Grant from City University of Hong Kong (7002480).

References

[1] A. Fisher, "World maze database," http://www.maze-world.com/, 2007.
[2] E. J. Morales, "Virtual mo," http://www.virtualmo.com, 2005.
[3] Conceptis Limited, "Conceptis puzzles," http://www.conceptispuzzles.com, 2007.
[4] G. Peatfield, "Maze creator," http://www.mazecreator.com/, 2007.
[5] H. Kern, *Through the Labyrinth: designs and meanings over 5000 years*. Prestel, 2000.
[6] C. Berg, "Amazeing art," http://www.amazeingart.com, 2007.
[7] H. K. Pedersen and K. Singh, "Organic labyrinths and mazes," in *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering*, 2006, pp. 79–86.
[8] J. Xu and C. S. Kaplan, "Image-guided maze construction," *ACM Transactoin on Graphics (SIGGRAPH'07)*, vol. 26, no. 3, pp. 29–37, 2007.
[9] A. Turing, "The chemical basis of morphogenesis," *Royal Society of London Philosophical Transactions Series B*, vol. 237, pp. 37–72, Aug 1952.
[10] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," in *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 1991, pp. 289–298.
[11] A. Witkin and M. Kass, "Reaction-diffusion textures," in *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 1991, pp. 299–308.
[12] L. O. Chua, *CNN: A Paradigm for Complexity*. World Scientific Publishing Company, 1998.
[13] C. S. Kaplan and R. Bosch, "TSP Art," in *Proceedings of Bridges 2005, Mathematical Connections in Art, Music and Science*, 2005, pp. 301–308.
[14] J. Xu and C. S. Kaplan, "Vortex maze construction," *Journal of Mathematics and the Arts I*, vol. 1, pp. 7–20, March 2007.
[15] H. Meinhardt, *Models of Biological Pattern Formation*. Prestel: Academic Press, 1982.
[16] H. Meinhardt, *The Algorithmic Beauty of Sea Shells*, 2nd ed. Springer, 1998.
[17] J. E. Pearson, "Complex patterns in a simple system," *Science*, vol. 261, pp. 189–192, 1993.
[18] A. Hagberg and E. Meron, "From labyrinthine patterns to spiral turbulence," *Physical Review Letters*, vol. 72, no. 15, 1994.
[19] L. Phan and C. Grimm, "Sketching reaction-diffusion texture," in *Eurographics Sketch Based Interfaces and Modeling workshop*. Eurographics, September 2006, pp. 107–114.
[20] T. Roska and L. O. Chua, "The CNN universal machine: An analogic array computer," *IEEE Transaction on Circuits and Systems II*, vol. 40, pp. 163–173, 1993.
[21] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.
[22] R. W. Zhou, C. Quek, and G. S. Ng, "A novel single-pass thinning algorithm and an effective set of performance criteria," *Pattern Recognition Letters*, vol. 16, no. 12, pp. 1267–1275, 1995.

[23] P. Selinger, "Potrace," http://potrace.sourceforge.net/, 2007.
[24] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.



Liang Wan received the B.Eng. and M.Eng. degrees in computer science and engineering from Northwestern Polytechnical University in China, in 2000 and 2003 respectively, and the PhD degree in computer science from the Chinese University of Hong Kong in 2007. She is currently a Research Fellow in the Department of Electronic Engineering, City University of Hong Kong. Her main research interest is computer graphics, including precomputed lighting, non-photorealistic rendering and GPU techniques.

Xiaopei Liu received his B.Eng degree in computer engineering from Huazhong University of Science and Technology in 2004. Currently, he is a Ph.D candidate in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. His main research interest lies in computer graphics, including image processing and analysis, surface modeling and rendering, simulation and visualizations.

Tien-Tsin Wong received the B.Sci., M.Phil., and PhD. degrees in computer science from the Chinese University of Hong Kong in 1992, 1994, and 1998, respectively. Currently, he is a Professor in the Department of Computer Science & Engineering, Chinese University of Hong Kong. His main research interest is computer graphics, including computational manga, image-based rendering, GPU techniques, natural phenomena modeling, and multimedia data compression. He received IEEE Transactions on Multimedia Prize Paper Award 2005 and Young Researcher Award 2004.

Chi-Sing Leung received the B.Sci. degree in electronics, the M.Phil. degree in information engineering, and the PhD. degree in computer science from the Chinese University of Hong Kong in 1989, 1991, and 1995, respectively. He is currently an Associate Professor in the Department of Electronic Engineering, City University of Hong Kong. His research interests include neural computing, data mining, and computer graphics. In 2005, he received the 2005 IEEE Transactions on Multimedia Prize Paper Award for his paper titled, "the Plenoptic Illumination Function" published in 2002. He is the Program Chair of ICONIP2009. He is also a governing board member of the Asian Pacific Neural Network Assembly (APNNA).