Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

Flow-Based Automatic Generation of Hybrid Picture Mazes

Fernando J. Wong and Shigeo Takahashi

The University of Tokyo

Abstract

A method for automatically generating a picture maze from two different images is introduced throughout this paper. The process begins with the extraction of salient contours and edge tangent flow information from the primary image in order to build the overall maze. Thus, mazes with passages flowing in the main edge directions and walls that effectively represent an abstract version of the primary image can be successfully created. Furthermore, our proposed approach makes possible the use of their solution path as a means of illustrating the main features of the secondary image, while attempting to keep its image motif concealed until the maze has been finally solved. The contour features and intensity of the secondary image are also incorporated into our method in order to determine the areas of the maze to be shaded by allowing the solution path to go through them. Moreover, an experiment has been conducted to confirm that solution paths can be successfully hidden from the participants in the mazes generated using our method.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications—J.5 [Computer Applications]: Arts and Humanities—

1. Introduction

History has been witness to the presence of mazes and labyrinths in ancient and modern human civilizations. From the Cretan and Egyptian labyrinths to the mazes found nowadays in assorted newspapers and weekly magazines, it is undeniable that mazes are, and have always been, a very close part of our culture. Mazes have also made their way into the artistic world in the works of Morales [Mor05], Christopher Berg [Ber08], and many others, thus turning mazes in conceptual tools for expressing feelings and ideas.

In this paper, we present an approach for automatically generating mazes from a given picture by taking into account its most important features and the direction of its edges. Although numerous maze generation programs are in existence, most of them create mazes based on square grids. This simple approach can instantly produce mazes of varied difficulty, however the expressive ability is rather limited and highly dependent on the skill of human artists. Our technical contribution here is to fully automate the maze generation process by incorporating the flow features inherent in the provided image motif, while keeping the associated mazes maximally aesthetic yet challenging.

We also focus our attention to a particular type of maze that has achieved high levels of popularity in recent years, called the *picture maze*, which has the special trait that an object or image is revealed once having solved the maze. Our second technical contribution lies in equipping our mazes with this trait in order to make them more attractive to maze solvers. To our knowledge, this is the first attempt to embed two different image motifs into a single maze, which we refer to as a *hybrid maze* throughout this paper. Figure 1 shows one of such mazes and its respective solution path.

This paper is organized as follows: Related work relevant to the proposed techniques is presented in Section 2. The details of our maze generation approach are explained in Section 3 while Section 4 describes our method for constructing the solution paths. Discussions and results of our method are presented in Section 5, followed by conclusions and pointers to possible future extensions given in Section 6.

2. Related Work

Maze design has been mainly pursued by artists like Berg and Morales and amateur maze designers such as Pullen [Pul08]. In particular, Pullen has gathered a huge collection of information on the creation and solution of mazes on his website Think Labyrinth [Pul08]. From algorithms

(a) Primary image (b) Secondary image (c) Resulting maze (d) Solution path

Figure 1: A hybrid maze generated with our approach.

and technical terms to psychological aspects, the information he has made available is very complete and comprehensive for those who wish to start or delve deeper into the world of maze design.

Pedersen and Singh [PS06] made research on the generation of organic labyrinth and maze structures. Their method is based on the evolution of a set of input curves into a labyrinth-like pattern, through the application of Brownian motion, fairing and attraction-repulsion forces. They extend this work to the creation of structured mazes as well but, as they point out, the nature of their approach is not really suitable for this purpose. Thus, their maze results considerably differ from those created by humans.

Probably, the most recent significant work on maze construction and design is the one introduced by Xu and Kaplan [XK07]. They designed a maze construction system in which the drawing area is divided into regions by using an input image as a guide. Then one of several maze textures is assigned to each of the regions, thus creating a set of independent mazes that would be then combined into a single one by breaking walls in the region boundaries. Such approach is recommended by Berg on his website Amazeing Art [Ber08], and can produce mazes of high quality. Although mazes generated with their system achieve high quality levels, ultimately, the maze construction process is highly manual, requiring the user to divide the area into regions, assign maze textures to each region, and adjust texture parameters as necessary.

On the other hand, none of the aforementioned designers and methodologies addresses the problem of image depiction through maze solution paths. Although both [PS06] and [XK07] allow the presence of user-specified solution paths in their mazes, their approaches are not really suitable for displaying images using the solution paths. Such problem is reminiscent of space-filling curves [DCOM00], single-strip triangulation of meshes [GE04] and artistic techniques such as continuous line illustration [BH04]. Worth mentioning is the work by Kaplan and Bosch [KB05] in which they propose a half-toning method based on a single line stroke obtained by solving an instance of the traveling salesman problem on a set of points obtained through image stippling. Unfortunately, such technique is not applicable to our case since proper image depiction usually depends on a large number of cities and their careful distribution.

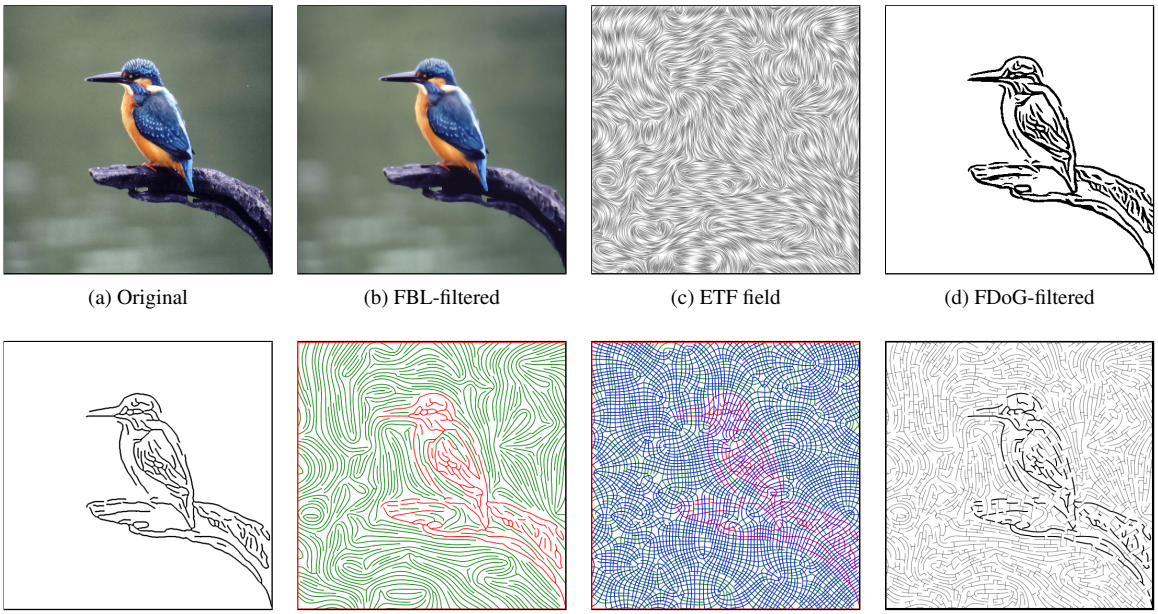
Our work on hybrid mazes is to some extent analogous to recent work on hybrid images [OTS06]. To create a hybrid image, two relatively similar images are combined into a single one after applying a low-pass filter to the first one and a high-pass filter to the second one. This results in images that seem to change as the viewing distance from the image varies. In our case, input images do not necessarily need to be similar to each other. However, visualization of the solution path image has been reported to improve as the distance from the maze is increased just as in hybrid images. In particular, our idea is probably more related to the picture mazes published by Conceptis Ltd. [Con08] in which a seemingly normal maze created from a square grid turns into an image once the solution path is traced.

3. Maze Grid Generation

As described earlier, our approach takes as input two different images, and builds a hybrid maze by automatically generating a maze grid oriented in the main edge directions of the primary image and then obtaining a path spanning the grid cells that correspond to the edges and dark regions of the secondary image. This sections delves on the first part, i.e., generation of the underlying maze grid.

Our approach for generating maze grids out of images is basically an extension of the work on directional mazes presented by Xu and Kaplan [XK07]. They create a maze by first inferring a vector field from several seed paths drawn by the user. Then, a set of evenly-spaced streamlines is traced in the direction of the vector field while a second one is traced perpendicular to the vector field, thus creating a grid. The maze is then created by removing walls from the grid.

We automate the construction of the grid by obtaining a vector field oriented according to the image edges. An edge-oriented vector field is usually obtained by rotating the image gradient perpendicularly [Her98]. However, this often results in vector fields lacking smoothness. Also, it has been demonstrated recently that an image edge field is better represented by a tensor field whose minor eigenvector field is colinear with the image gradient [ZHT07]. Nonetheless, for



(a) Original (b) FBL-filtered (c) ETF field (d) FDoG-filtered

(e) Line thinning (f) Primary streamlines (g) Maze grid (h) Resulting maze

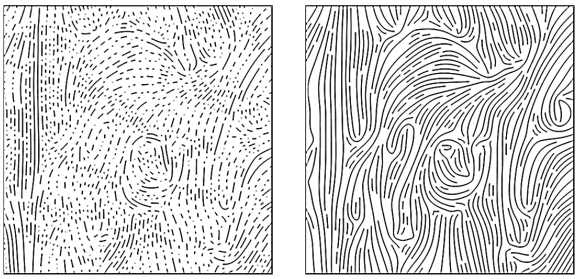
Figure 2: Overview of the maze grid generation process: (a) An input image is processed with (b) the FBL filter. (c) A heavily smoothed ETF field is then obtained from the filtered image. (d) The FDoG filter is applied to the FBL-filtered image and (e) the output is thinned. (f) A primary set of streamlines (in green) is traced in the direction of the vector field by taking the extracted edges (in red) as a reference. (g) A secondary set of streamlines (in blue) is then traced perpendicular to the vector field direction, thus creating a grid. (g) A maze can be generated by removing walls from the grid.

our purpose, we have found that the Edge Tangent Flow (ETF) field proposed in [KLC09] provides fairly smooth vector fields in a reasonable amount of time. We also make use of the Flow-based Bilateral (FBL) and Flow-based Difference of Gaussians (FDoG) filters proposed in [KLC09] as well. Also, advanced techniques for generating highly uniform quad meshes from vector fields have been proposed [RLL*06]. However, we have opted for slightly modifying the streamline-based approach of [XK07], since this is a fairly simple approach and results are good enough for our purposes.

A grid is generated from the input image (Figure 2a) by first filtering it through the FBL (Figure 2b) in order to smooth the image regions while sharpening the edges. A vector field oriented in the edge directions is then obtained by applying the ETF operator (Figure 2c). The most significant edges of the image are then extracted with the FDoG filter (Figure 2d). In the final maze, walls derived from these contours are drawn thicker than others in order to provide better depiction. In our implementation, we apply 100 iterations of the ETF and 5 iterations of both the FBL and FDoG filters, all of them with the default parameters specified in [KLC09]. We use a line thinning algorithm to obtain rather uniform line segments from the extracted edges (Figure 2e). We have chosen the line thinning algorithm proposed in [HWL03] for its simplicity and quality of results. Additionally, cubic spline interpolation is applied to the thinned line segments in order to smooth them. Streamlines are then traced by taking these thinned line segments as a reference (Figure 2f). In the same way as Xu and Kaplan, we use the streamline placement algorithm proposed by Jobard and Lefer [JL97]. The maze grid is finished with the tracing of streamlines perpendicular to the vector field (Figure 2g).

Unlike vector fields used in [XK07], ETF fields usually present a high amount of singularities, often resulting in rather short streamlines (Figure 3a). Fortunately, the ETF operator produces vector fields with a high degree of smoothness around their singularities. Thus, instead of stopping the growth of a streamline once a singularity is found, we can continue growing it beyond the singularity, in direction opposite to the vector field (or in the field direction if it was initially being grown opposite to it) as shown in Figure 3b. This effectively reduces the number of short streamlines.

Once a grid has been created, a maze can be constructed by just removing some walls (Figure 2h). We bias our mazes so that passages tend to flow in the direction of the vector field. Biasing consists on assigning relatively larger weights



(a) (b)

Figure 3: (a) Excess of short line segments due to large presence of singularities. (b) This can be overcome by continuing the growth of streamlines beyond singularities.

to maze walls parallel to the vector field and smaller weights to walls perpendicular to it, so that walls perpendicular to the vector field have a tendency to being processed first. In our implementation, we accomplished this by choosing real numbers r, s, t, and u in such a way that $r < s < t < u$, and assigning random weights within the range $[r, t]$ to perpendicular walls and weights within the range $[s, u]$ to all other walls. We refer the readers to [XK07] for more information on this maze biasing.

4. Feature-based Solution Paths

4.1. Basic Strategy

Given a previously generated image-derived grid, we obtain a solution path from a secondary image (Figure 4a) as follows:

- Extract the most significant contour edges of the image.
- Determine which cells can be used as part of the solution path based on the extracted contours and color intensity information.
- Fill the corresponding cells with cyclic paths.
- Combine all cyclic paths into a single cycle.
- Modify the remaining cycle to obtain a single path with specified starting and ending points.

The problem of finding a solution path that represents the main features of an image is NP hard by itself. However, this can be approximated in a visually-plausible manner with our approach.

4.1.1. Extraction of Contour Features

Edge detection is performed in the same way as in the previous section on maze grid generation. We apply the FBL, followed by the FDoG filter to the secondary image to extract its most significant edges and proceed to apply line thinning on them (Figure 4b). Users are optionally allowed to freely modify these contours, in order to provide more detail while reducing the number of unnecessary features.

4.1.2. Obtaining the Set of Potential Solution Path Cells

Before attempting to find the solution path itself, we need to specify which cells can be actually part of this path. We start by inferring a set of cell sequences from the extracted contours (Figure 4c). A *cell sequence* is an ordering of grid cells such that each cell is a neighbor of the next one and no cell is duplicated in the sequence. For each contour, we identify the cells it intersects on the grid and create a sequence based on the order these cells were intersected. It is not particularly an issue if cells belonging to more than one sequence exist, as they will only be used as a base for deciding which cells can be part of the solution path.

A solution path representing the main contour features of the secondary image could be generated based on the obtained cell sequences alone. However, this approach often leads to easily noticeable solution paths due to perturbations in the flow of passages, as shown in Figure 5. In order to avoid this, we have opted for representing the image contours as if they had some degree of thickness. This approach effectively allows the representation of the contours, while leaving enough space to trace the solution path in a way that reduces the amount of passage perturbation. For this reason, we also label, as potential members of the solution path, all cells separated by at most two cells from any of the previously obtained sequences (Figure 4d).

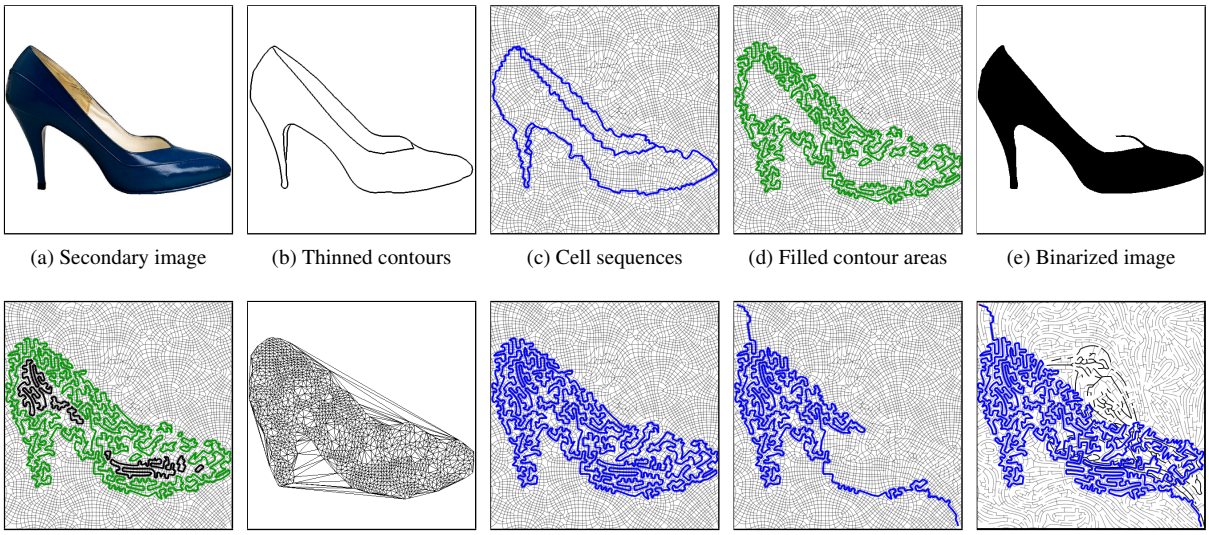
The current set of labeled cells partitions the remaining grid cells into several regions. We can use some of these cells to represent shadows or areas of low color intensity in the image. This can be accomplished by generating a binarized version of the secondary image (Figure 4e) and labeling the cells that correspond to its black areas (Figure 4f). We can use simple binary thresholding for this or more advanced thresholding techniques such as the one proposed in [XK08]. We allow users to edit the binarized version of the image as well in order to provide more control over the parts of the maze that will be covered with the solution path.

4.1.3. Filling Maze Cells

Knowing the areas of the grid on which the solution path can go through, we proceed to fill those areas with cycles while avoiding passages that disrupt the flow of the maze as much as possible.

We can easily fill with cycles a given region by taking the following steps:

1. Choose any pair of unprocessed adjacent cells a and b in the region.
2. Create an initial cycle with edges (a,b) and (b,a) (Figure 6a).
3. Select an edge (p,q) from the cycle.
4. Attempt to grow the cycle from (p,q) by assimilating the cells neighboring p and q, while retaining the cycle condition (Figure 6b).



(a) Secondary image (b) Thinned contours (c) Cell sequences (d) Filled contour areas (e) Binarized image

(f) Filled regions (g) Delaunay triangulation (h) Resulting cycle (i) Custom endpoints (j) Resulting maze

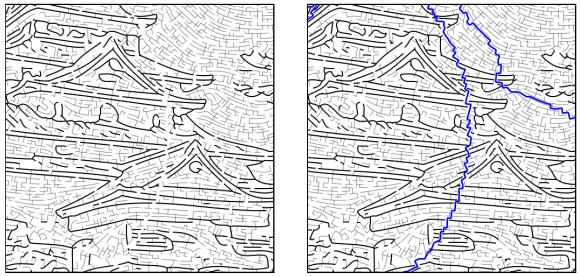
Figure 4: Overview of the solution path generation process: (a) A secondary image has its contours extracted (b). (c) Cell sequences are found by intersecting the thinned edges with a previously generated grid. (d) The vicinity of the sequences is filled with cycles. (e) A binarized version of the secondary image is obtained. (f) Cells corresponding to dark areas of the binarized image are also filled with cycles. (g) The Delaunay triangulation of the centroids of all cells belonging to any of the cycles is computed. (h) The disjoint cycles are then combined into a single one in a shortest distance basis. (i) Paths P_s and P_e are respectively found from the specified starting and ending points to any cells in the cycle, and the cycle is opened at these cells, creating a path P_l spanning from start to end by appending P_s and P_e to the longest path derived from the split cycle. (j) The resulting solution path is generated by filling the corresponding regions with cycles and merging them with P_l.

5. Repeat Steps 3-4 until no more cells can be assimilated into the cycle.
6. Repeat Steps 1-5 until no pairs of unprocessed adjacent cells are left in the region (Figure 6e).

This process effectively generates a set of disjoint cycles that span the specified region, but these cycles very often disrupt the flow of passages in the resulting maze. We can reduce the amount of disruptions by making use of the biasing technique, explained earlier in Section 3, in the following manner:

- Denote as *parallel edges* all edges in the cycle whose endpoints correspond to grid cells separated from each other by a wall oriented perpendicular to the vector field.
- Bias edge selection in Step 3, so that parallel edges are less likely to be chosen than other edges (Figure 6c).
- Further bias edge selection, so that any edges newly created by growing the cycle from a parallel edge at Step 4 are chosen before any other edges (Figure 6d).

Using this algorithm, we start by filling the areas corresponding to the contours of the image and then proceed to fill each region separately.

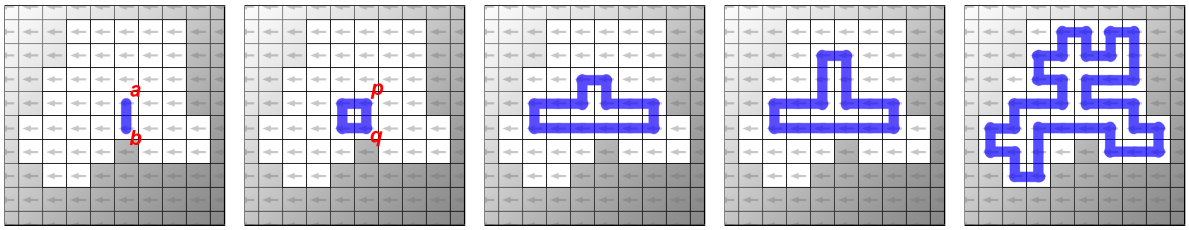


(a) (b)

Figure 5: (a) The solution path can be clearly seen going from top to bottom in this maze. (b) This often occurs when we attempt to find solution paths based only on the cells intersected by the extracted contours.

4.1.4. Merging of Cyclic Paths

The merging of cyclic paths is rather simple, since the union of two cycles always results in another cycle being created. We would like to merge these cycles in a shortest distance basis in the following way:



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

Figure 6: Growth of a cycle in a horizontally biased region: (a) A cycle is created from a random edge placed in the region. (b) The cycle is then grown from an edge by appending neighboring cells. (c) The algorithm is biased so that growth is less likely to occur from edges oriented parallel to the vector field. (d) Once such an edge is processed, we further bias the algorithm so that edges created at this step are considered before any other edges, in order to avoid passage distortions. (e) The process is repeated iteratively until no more cells can be assimilated. Note that we focus on a local square-pattern grid in this figure. We may have a non-regular grid pattern in our framework.

1. Calculate the set C of the centroids of all cells belonging to any cycle in the region.
2. Compute the Delaunay triangulation T of C (Figure 4g).
3. Let $g(x)$ denote the grid cell corresponding to centroid $x \in C$.
4. Remove the shortest edge $(u,v) \in T$.
5. If $g(u)$ and $g(v)$ are in different cycles, find $P(g(u),g(v))$, the shortest path between $g(u)$ and $g(v)$.
6. If $P(g(u),g(v))$ exists, find another path $P'(g(u),g(v))$ that joins both cycles while keeping close to $P(g(u),g(v))$. This can be done by restricting the shortest path algorithm to the cells separated by at most 2 cells from any cell in $P(g(u),g(v))$ and applying distance penalties each time a cell not in the direct neighborhood of $P(g(u),g(v))$ is chosen as part of the path.
7. If $P'(g(u),g(v))$ exists, open both cycles at the cells corresponding to the endpoints of $P(g(u),g(v))$ and $P'(g(u),g(v))$, and merge them by using $P(g(u),g(v))$ and $P'(g(u),g(v))$.
8. Repeat Steps 4-7 until no more edges are left in T or only one cycle remains over the domain of grid cells (Figure 4h).

This procedure is similar to finding the Euclidean minimum spanning tree of a set of points while taking initial point clusters into account. As in the previous subsection, we start merging the cycles corresponding to the contours before merging the cycles in the regions.

4.1.5. Arbitrary Starting and Ending Points

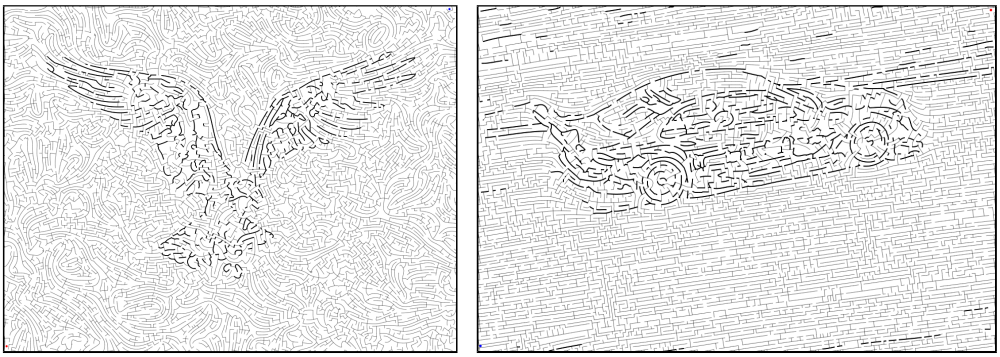
The cycle obtained through the algorithm explained in the previous subsection can easily serve as the solution path of the maze, as it illustrates the shapes of the input image. However, the starting and ending points of such a path would always be relatively close to each other. Nevertheless, we can easily modify this cycle in order to obtain a path with arbitrary endpoints:

1. Find a path P_s from the specified starting point s to any cell c in the cycle.
2. Find a path P_e from the ending point e to any cell d in the cycle.
3. Split the cycle in two paths by breaking it at cells c and d. Denote the longer of the two as P_l and discard the shorter one.
4. Append P_s and P_e at the front and back of P_l (Figure 4i).
5. Assume a hidden edge (s,e) exists, turning P_l into a cycle.
6. Apply the algorithm explained in Section 4.1.3 by taking P_l as an initial cycle. Edge (s,e) must be excluded during the process, so that P_l is never grown from such an edge.
7. Merge the resulting cycles by using the algorithm mentioned in Section 4.1.4 (Figure 4j).
8. Remove edge (s,e).

This simple algorithm results in a single path spanning from the specified starting and ending points of the maze. Although we have fixed the endpoints of the solution paths at the top-right and bottom-left corners of our examples, we can virtually have any combination of endpoints. Once the solution path is found, the maze can be finished by removing the walls intersected by the path and breaking the remaining walls of the maze as necessary.

4.2. Miscellaneous Issues

We have noticed that the approach described in this section works well for rather simple secondary images with well defined edges, such as cartoon pictures. It performs better if the image in question has only one object in it. We do not see this as a serious constraint as current picture mazes usually depict a single object once solved. Quality is also dependent on the amount of cells in the grid. Mazes generated through fine sampling of the image flow field usually achieve solution paths with better depiction results. Also, in order to



(a) Bird (b) Racing car

(c) Jet fighter (d) Taj Mahal

Figure 7: Mazes created for the first experiment.

Table 1: Step-by-step timing results of maze construction.

Maze	Size	FBL	ETF	FDoG	LT	SP	CS	RF	CM	CE	Total
Bird + Hat	1024×782	5.953	37.490	3.401	0.307	44.230	0.378	2.761	1.035	2.314	97.869
Car + Wine glasses	1024×683	5.111	26.998	2.848	0.361	27.137	0.670	12.892	1.305	14.689	92.012
Jet Fighter + Shark	1024×681	5.126	29.902	2.880	0.250	22.228	0.360	0.645	0.952	0.624	62.967
Taj Mahal + Flower	1024×844	6.446	38.629	3.670	0.493	58.323	0.641	3.469	1.376	2.069	115.115

FBL: Flow-based Bilateral **ETF:** Edge Tangent Flow **FDoG:** Flow-based Difference of Gaussians **LT:** Line Thinning
SP: Streamline Placement **CS:** Cell Sequences **RF:** Region Filling **CM:** Cycle Merging **CE:** Custom Endpoints
All measurements are in seconds.

avoid possible distortions in the flow of passages, it is recommended to use a secondary image with edges that tend to flow in the direction of the underlying vector field.

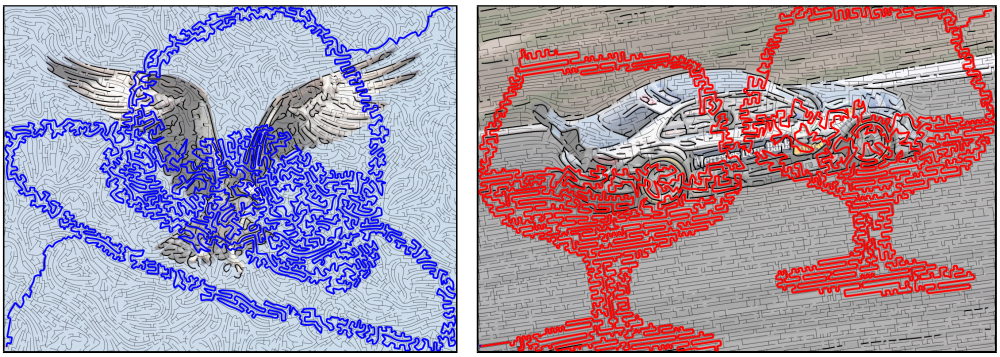
5. Discussions and Results

5.1. Implementation Details

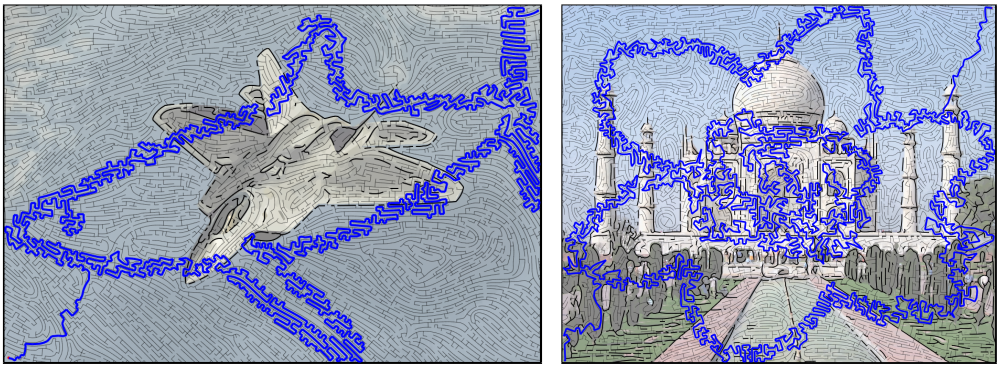
Several results of our approach can be seen in Figures 1, 7, and 8. The CGAL library [CGA07] was used for handling grids and calculating Delaunay triangulations in our mazes. Creating a maze from 1024×768 images usually takes around 90 to 100 seconds on an Intel Core 2 Duo E6550 2.33 Ghz CPU with 2 GB of RAM. Table 1 provides step by step timing data for generating the mazes shown on Figure 7.

5.2. Recognition of Image Motifs in Mazes

Two experiments were conducted in order to verify the effectiveness of our approach. The first experiment consisted on showing hybrid mazes (Figure 7) to 100 participants and asking if they could identify the primary image from the maze walls and also guess the shape of the hidden object without solving the puzzle. A second experiment with the same mazes, this time with color information and drawn solution paths (Figure 8), was conducted on the same group. Again, they were asked if they could recognize the primary



(a) Bird (b) Racing car



(c) Jet fighter (d) Taj Mahal

Figure 8: Mazes created for the second experiment.

Table 2: Results of the first experiment.

Maze (Primary + Secondary Images)	PI	SI
Bird + Hat	100/100	0/100
Car + Wine glasses	100/100	1/100
Jet Fighter + Shark	92/100	1/100
Taj Mahal + Flower	99/100	0/100

PI: Subjects able to recognize the primary images.
SI: Subjects able to guess the secondary images.

Table 3: Results of the second experiment.

Maze (Primary + Secondary Images)	PI	SI
Bird + Hat	100/100	89/100
Car + Wine Glasses	100/100	100/100
Jet Fighter + Shark	99/100	100/100
Taj Mahal + Flower	100/100	99/100

PI: Subjects able to recognize the primary images.
SI: Subjects able to identify the secondary images.

image and if they could perceive which kind of object the secondary image was. Tables 2 and 3 summarize the results of these experiments.

Our results show that color information is not a critical factor for depiction depending on the primary image used on our mazes. Also, they show the importance of proper image selection. In particular, the edges and color intensity of the input images should be carefully evaluated. For example, the hat in Figure 10a has dark areas on its sides. The binarized version of the image inherited these dark areas, resulting in a large filled region in the middle of the maze (Figure 8a), which was reported to interfere with the perception of the image by the participants. This can, however, be simply addressed by removing those areas from the binarized image. Moreover, the shape and orientation of this object makes it easy to mistake it with other similar objects (such as an UFO) when turned into a solution path.

As shown in the results, very few of the participants were able to identify the secondary images without solving the mazes first, thus proving the effectiveness of our approach in keeping the solution paths hidden while still providing good representation of the image motifs.

5.3. Maze Difficulty

Although results show that our mazes can effectively represent both images while hiding the solution path, feedback

from the participants pointed at the difficulty in solving our mazes.

The degree of difficulty in our mazes is highly dependent on three factors:

- Solution path length: Mazes become more difficult as the solution path increases in length.
- Length of dummy branches: Mazes with longer paths leading to dead ends are generally more difficult to solve.
- Amount of available grid cells: The difficulty of a maze increases proportionally to the amount of cells in the grid.

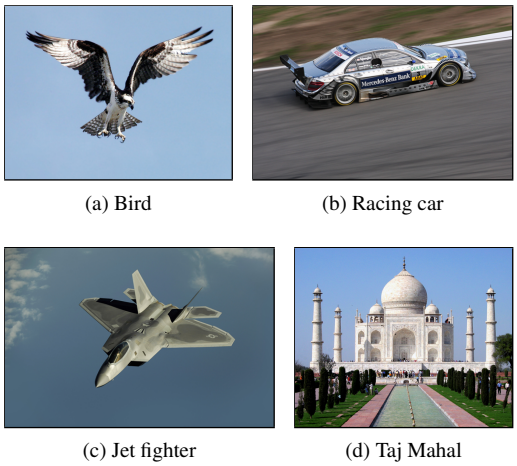
Since our approach requires long paths to effectively conceal the solution of the maze, reducing the difficulty of mazes by shortening the length of the solution path is often a very prohibitive option. However, we can still provide control in the difficulty of our mazes by manipulating the other two factors to some extent.

Our prototype system can limit the length and branching of dummy paths stemming from the solution path. We essentially grow passages, of a random length not exceeding the specified maximum, at several random locations on the solution path, transforming it into a tree. These paths are randomly generated, but biased to flow mostly in the direction of the underlying field. Another parameter controls the amount of branching levels allowed on these paths. After applying this process, the remaining cells of the maze can either be connected to the tree at random points or isolated from the tree itself, thus generating a set of disjoint mazes and limiting the amount of cells that maze solvers have access to. We choose to do the latter, since the former could still produce dummy paths of undesirable lengths. This idea is still in an early stage, and more sophisticated methods for further reducing the degree of difficulty is left as a topic for future work.

Note that the mazes displayed in Figures 7 and 8 were actually generated with controlled difficulty as explained above. These mazes were used in a third experiment on which 10 participants were asked to actually solve them while tracking the time spent in solving this type of mazes. Figure 11 provides the gathered timing data for this experiment. It appears that, on average, one of our mazes should take between 15 to 25 minutes to solve. However, this is with controlled maze difficulty as we had explained before. Notice that this is about the same amount of time required to solve a regular picture maze, which are, in general, much smaller in size.

5.4. Degree of Solution Path Perturbation

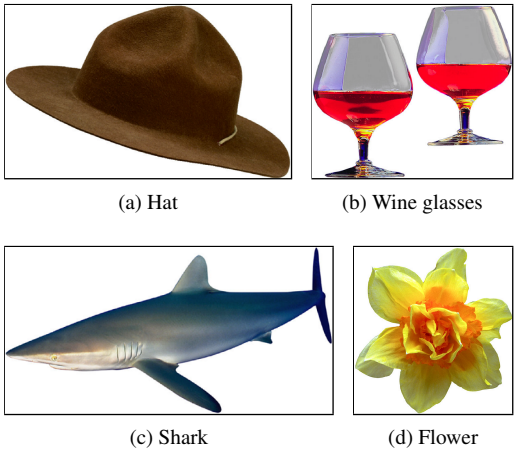
A fourth experiment was conducted in order to determine the amount of time required to solve a maze depending on the degree of perturbation of the solution path. As you might recall, our solution paths represent the edges of the secondary image as if they had some degree of thickness. The same 10 participants were asked to solve a set of three mazes generated with the same combination of primary and secondary images, but having solution paths with different "thickness" for representing the contours: a single contour solution path, a slightly thicker solution path and an overly thick solution path. Figure 12 summarizes the time required to solve these mazes by each of the participants.



(a) Bird (b) Racing car

(c) Jet fighter (d) Taj Mahal

Figure 9: Primary images used to generate the maze grids.



(a) Hat (b) Wine glasses

(c) Shark (d) Flower

Figure 10: Secondary images used to create the solution paths.

6. Conclusion and Future Work

An automated method for building mazes from images has been presented in this paper by taking advantage of recently developed techniques for edge detection and region smoothing of images. Moreover, this work has been further extended by allowing the customization of the maze solution

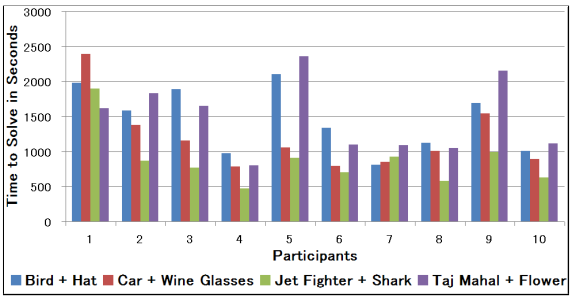


Figure 11: Results of the third experiment.

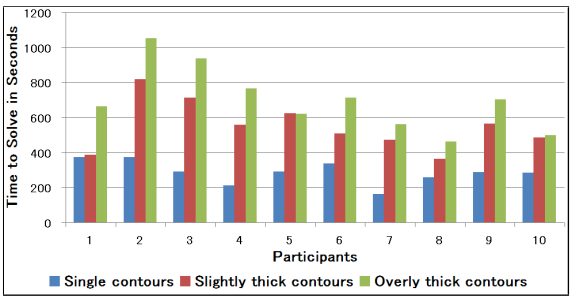


Figure 12: Results of the fourth experiment.

paths to portray the features of an additionally provided image. The experimental results have shown the feasibility of our approach in the depiction of images with hybrid mazes. Discussion on the proper selection of images has also been provided.

Although our results achieve high visual quality for a large group of images, they still lack the level of detail found on the mazes of Christopher Berg and other maze designers. The maze generation process introduced here could be incorporated into Xu and Kaplan's system [XK07] as a type of additional feature-based texture, possibly helping designers into creating mazes that take into account the topology of the input image. Our solution path generation process could be incorporated into other maze construction approaches as well, as long as they can provide a previously generated grid. Another possible extension would be the establishment of a measure of compatible images for hybrid mazes to automate the image selection process. We also would like to explore the applicability of the present approaches to other problems, such as the rendering of image mosaics [EW03] and continuous line illustrations [KB05].

Acknowledgements

This work has been partially supported by Japan Society of the Promotion of Science under Grants-in-Aid for Scientific Research (B) No. 20300033.

References

[Ber08] BERG C.: Amazeing art, 2008. http://www.amazeingart.com/.

[BH04] BOSCH R., HERMAN A.: Continuous line drawings via the traveling salesman problem. *Operations Research Letters 32*, 4 (2004), 302–303.

[CGA07] CGAL EDITORIAL BOARD: *CGAL User and Reference Manual*, 3.3 ed., 2007.

[Con08] CONCEPTIS LTD.: Conceptis puzzles, 2008. http://www.conceptispuzzles.com/.

[DCOM00] DAFNER R., COHEN-OR D., MATIAS Y.: Context-based space filling curves. *Computer Graphics Forum 19*, 3 (2000), 209–218.

[EW03] ELBER G., WOLBERG G.: Rendering traditional mosaics. *The Visual Computer 19*, 1 (2003), 67–78.

[GE04] GOPI M., EPPSTEIN D.: Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum 23*, 3 (2004), 371–379.

[Her98] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of SIGGRAPH '98* (1998), pp. 453–460.

[HWL03] HUANG L., WAN G., LIU C.: An improved parallel thinning algorithm. In *ICDAR '03: Proceedings of the 7th International Conference on Document Analysis and Recognition* (2003), IEEE Computer Society, pp. 780–783.

[JL97] JOBARD B., LEFER W.: Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing* (1997), pp. 45–55.

[KB05] KAPLAN C. S., BOSCH R.: TSP Art. In *Proceedings of Bridges 2005, Mathematical Connections in Art, Music and Science* (2005), pp. 301–308.

[KLC09] KANG H., LEE S., CHUI C. K.: Flow-based image abstraction. *IEEE Transactions on Visualization and Computer Graphics 15*, 1 (2009), 62–76.

[Mor05] MORALES J. E.: Virtual Mo, 2005. http://www.virtualmo.com/.

[OTS06] OLIVA A., TORRALBA A., SCHYNS P. G.: Hybrid images. *ACM Transactions on Graphics 25*, 3 (2006), 527–532.

[PS06] PEDERSEN H., SINGH K.: Organic labyrinths and mazes. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (2006), pp. 79–86.

[Pul08] PULLEN W. D.: Think labyrinth, 2008. http://www.astrolog.org/labyrnth.htm.

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Transactions on Graphics 25*, 4 (2006), 1460–1485.

[XK07] XU J., KAPLAN C. S.: Image-guided maze construction. *ACM Transactions on Graphics 26*, 3 (2007), 29.

[XK08] XU J., KAPLAN C. S.: Artistic thresholding. In *NPAR '08: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering* (2008), pp. 39–47.

[ZHT07] ZHANG E., HAYS J., TURK G.: Interactive tensor field design and visualization on surfaces. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (2007), 94–107.