Image-Guided Maze Construction

Jie Xu Craig S. Kaplan
Computer Graphics Lab
David R. Cheriton School of Computer Science
University of Waterloo

Abstract

We present a set of graphical and combinatorial algorithms for designing mazes based on images. The designer traces regions of interest in an image and annotates the regions with style parameters. They can optionally specify a solution path, which provides a rough guide for laying out the maze's actual solution. The system uses novel extensions to well-known maze construction algorithms to build mazes that approximate the tone of the source image, express the desired style in each region, and conform to the user's solution path.

CR Categories: I.3.m [Computing Methodologies]: Computer Graphics—Miscellaneous J.5 [Computer Applications]: Arts and Humanities

Keywords: maze, labyrinth, streamline, halftoning, line drawing

1 Introduction

Mazes and labyrinths have enjoyed a long, venerable tradition in the history of art and design. They have been used as pure visual art, as architectural decoration, and as cultural and religious artifacts. They are a frequent source of metaphor in art and in life. They have been executed in media from pencil to topiary, and range in size from tiny to more than twenty acres. Mazes have always been popular as printed puzzles. Their popularity is also growing in level design for computer games, and as walk-through puzzles in cornfields. Maze designer Adrian Fisher claims that we are currently living in a golden age of mazes [Fisher 2006].

The different artistic or spiritual motivations behind the construction of mazes and labyrinths lead to a wide range of visual styles. The eleven-circuit labyrinth on the floor of the Chartres cathedral is abstract and geometric; its single long path is intended as a journey of meditation and penitence. Other designs are delicate works of representational art, line drawings that are simultaneously puzzles and stylized depictions of real-world scenes. Figure 2(a) shows the work of sixteenth century Paduan architect Francesco Segala [Kern 2000].

We are especially inspired by the work of Christopher Berg, a contemporary archaeologist-turned-maze designer. He draws stunning mazes based on monuments and ancient wonders [Berg 2001], as shown in Figure 2(b). On his website, Berg offers some insighful observations on the construction of mazes [Berg 2005]. Of course, his analysis is intended for artists and does not translate directly into a computer implementation.

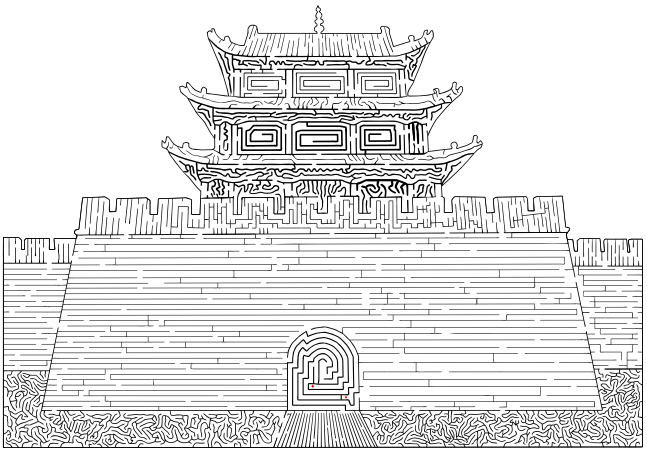


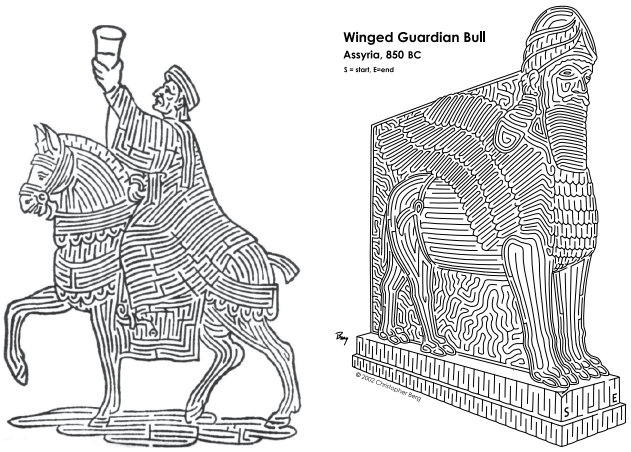
Figure 1: An example of a maze created using the technique described in this paper. The maze is based on a photograph of the Jia Yu Guan pass of the Great Wall of China.

We are interested in the problem of automatically synthesizing pictorial mazes from images. In particular we would like the computer to handle the details of placing individual passages and walls, freeing the designer to make high level decisions about artistic intent, style, texture, and layout. In this sense, mazes seem to overlap with other problems in non-photorealistic depiction, most notably pen-and-ink illustration [Winkenbach and Salesin 1994]. There, strokes must simultaneously convey tone, texture, and form. We are faced with the same issues, plus an additional *topological* challenge: the spaces between the strokes must function as a maze.

In this paper, we present a set of graphical and combinatorial algorithms that support the construction of mazes resembling user-supplied images. We have also developed an interactive application that lets a designer author a maze at a high level. An example produced with our system is shown in Figure 1.

In our system the designer manually partitions an image into a set of regions and assigns style parameters to each region (Section 4). They are also able to sketch a schematic layout for the maze's solution path (Section 5). The final maze will contain intertwined passages in each region, connected together by breaking walls between regions. Berg discusses a similar two-level approach to maze construction, stating that at a high level the maze should be seen as a set of areas connected by openings called "bottlenecks" [Berg 2005].

Our technique produces a maze that is drawn in the designer's choice of styles, has the connectivity suggested by the solution path, and reproduces the tone of the original image (Section 6.1). We also support a limited amount of foreshortening (Section 6.2) to help convey perspective and shape.



(a) (b)

Figure 2: A sixteenth-century woodcut by Francesco Segala, and a maze by Christopher Berg depicting an Assyrian winged bull.

2 Related work

There are many patterns in nature that resemble mazes, such as coral and the folds of the brain's cortex. Stevens [1974] shows several examples, which he classifies as "meanders". Given the right parameters, Turk's reaction-diffusion textures can produce maze-like designs [Turk 1991], but the results are image-based and difficult to control in the context of this work. A cellular model similar to Turk's was proposed in the biophysical literature to simulate the behaviour of cholesterol molecules [Huang and Feigenson 1999].

Programs for generating mazes are everywhere; there is no shortage of programmers who demonstrate their skills with online maze generators. Yet we can find very little research or professional design software that deals with maze generation. Even Berg draws a skeleton of lines by hand, scans in the drawing, and adds in the dead ends manually with a vector drawing tool.

Walter Pullen, an amateur maze designer, is the author of the fully-featured program Daedalus [Pullen 2005]. His "Think Labyrinth" website catalogues an enormous variety of maze styles and algorithms for maze generation. It is the most comprehensive resource we know of for computer-generated mazes. One other piece of software we are aware of is Peatfield's MazeCreator [Peatfield 2005]. This commercial tool can produce mazes resembling real-world imagery, but it does so in a naïve way. Peatfield approximates the image with a square grid, and constructs a maze in the grid. The results lack the grace and appeal of the freeform designs by Segala and Berg.

In 2004, the puzzle company Conceptis Limited introduced a related puzzle they call Maze-a-Pix [Conceptis Limited 2006]. Initially, a Maze-a-Pix resembles an ordinary rectangular maze. However, the cells that make up the solution path, when filled in, reveal a hidden image. Interestingly, the design of Maze-a-Pix puzzles is intractable (it is a variation of the Hamiltonian path problem).

Xu and Kaplan have recently demonstrated a technique for drawing abstract geometric mazes based on arrangements of vortices [Xu and Kaplan 2007]. A vortex is a common obfuscating device in maze design in which multiple spiral arms converge on a common centre. The goal of their work is not stylized depiction. Their mazes are abstract and geometric, and although they include ideas for increasing the appeal of their results, the overall aesthetic range is limited. Their work is primarily an attempt to understand the notion of difficulty in mazes by constructing hard-to-solve examples.

In the computer science literature, the most relevant previous work is that of Singh and Pedersen [2006]. They present a geometric attraction-repulsion model that evolves an initially simple shape into an organic labyrinthine drawing. A set of spatially-varying parameters control features such as line density and anisotropy. Their work is focused almost entirely on the construction of labyrinths (single paths with no branches). They point out that their labyrinths can be turned into mazes, but the results do not resemble human-designed mazes. Still, their algorithm can play an important role in maze construction (see Section 4.3). Kaplan and Bosch's TSP Art [2005] achieves a related visual style by finding a low-cost TSP tour over the points in a stippled drawing. The "labyrinthine portraits" drawn manually by Morales [2006] are also strikingly similar.

3 Maze basics

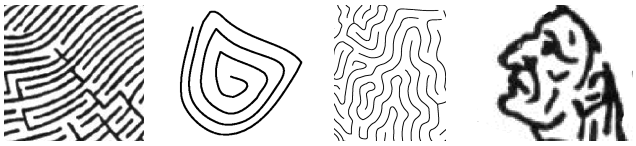
The construction of simple mazes is a classic exercise in computer programming. In this section, we review the traditional approaches to automated maze construction. We also introduce terminology for the geometric objects and data structures that will be used in the rest of the paper.

We begin with a planar subdivision, an embedding of a planar graph in which we can identify vertices, edges, and faces. We refer to this subdivision as a *grid* and to its faces as *cells*. When two adjacent cells intersect in a connected set of edges, we refer to the edges collectively as a *cell wall*. We build a maze by erasing some of the cell walls. When there is a path between any pair of cells that does not cross a wall, the maze is *connected*. When each of these paths is unique then the maze contains no cycles and is called *perfect*. In this paper we consider only perfect mazes, though most of the techniques apply equally well to mazes with cycles.

It is sometimes helpful to consider the dual of the grid, which we call the *cell graph*. Every cell in the grid is a vertex in the cell graph, and two vertices in the cell graph are connected by an edge if their corresponding cells have a wall in common.

With the dual viewpoint, a perfect maze on a grid can easily be seen to correspond to a spanning tree of its cell graph. Most maze construction techniques are therefore nothing more than algorithms for constructing random spanning trees. For example, a standard approach described by Shivers [2005] is just Kruskal's algorithm [Cormen et al. 1992]. In Kruskal's algorithm, we iterate over the cell walls; a wall is deleted (or, equivalently, an edge in the cell graph is added to a spanning tree) provided that doing so would not create a cycle in the maze. Breaking a wall would create a cycle if the cells on either side of the wall are already connected via a path. Sets of connected cells are maintained using a union-find data structure.

Pullen also shows how wall-breaking can be "biased", so that some walls are more likely to be broken than others. We can assign weights to cell walls and visit them in increasing order of weight, taking advantange of the fact that we are free in Kruskal's algorithm to visit walls in any order. Walls considered earlier are more likely to be broken. For example, to bias maze construction in a rectangular grid to prefer vertical passages, we choose real numbers $0 < a < b < 1$, assign horizontal walls weights chosen uniformly from the interval $[0,b]$, and vertical walls weights from $[a,1]$. Horizontal walls are therefore more likely to be deleted first, leaving behind more vertical passages. The amount of overlap between the two intervals determines how strong the bias is. We use variations on this technique in the textures of the following section.



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

Figure 3: Excerpts from existing mazes showing the textures we support in this work: a directional region (a), a spiral region (b), a random region (c), and user-defined lines (d). All examples are taken from Figure 2.

Our system is based on the idea of dividing an object up into pieces, constructing a fragment of a maze in each piece, and connecting the fragments together to form a complete maze. The designer traces an outline over an image that defines the extent of the maze; we call this outline the maze's *footprint*. The designer then segments the footprint into *regions*. We do not automate the segmentation, but provide Intelligent Scissors [Mortensen and Barrett 1995] to facilitate the drawing of boundaries.

4 Maze textures

The core means of artistic expression in our technique is the ability to assign maze textures on a region-by-region basis. The range of possible textures is unbounded; indeed, Singh and Pedersen [2006] show how to interpolate continuously between parameters defining their styles. We limit ourselves to a few choices for a region's style, motivated by our survey of the artform (see Figure 3). We concentrate on four textures: directional mazes, spiral mazes, random mazes, and user-defined lines. Each texture is controlled by specialized parameters.

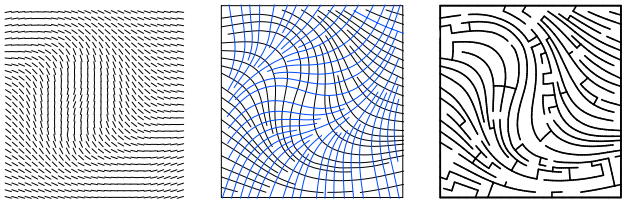
A key requirement for our maze textures is the ability to control the spacing between adjacent walls, because the local density of walls can be used to depict tone. In this section we discuss how to achieve a desired local spacing of walls in each maze texture; in Section 6.1 we then show how to reproduce the tone of the original image by controlling the widths of the walls as well as their spacing.

A maze texture is a procedural grid generator. The grid produced by the texture is then passed to the routing algorithm of Section 5 to be turned into a maze. The random texture of Section 4.3 is an exception in that most of the appearance is generated in a post-process after the maze is constructed.

4.1 Directional mazes

We would like to construct mazes with an overall anisotropy, so that the maze's walls tend to flow in a prescribed direction. A good example of this sort of flow is Ostromoukhov's work on facial engraving [Ostromoukhov 1999], though it would be difficult to achieve local tone control with his warped square grids. We take inspiration from the pen-and-ink work of Salisbury et al. [1997], where line placement is guided by a user-defined vector field.

In our system, the designer draws any number of seed paths in a given region, and the system infers a vector field using interpolation from radial basis functions [Hays and Essa 2004]. We can then see the line placement problem as equivalent to streamline visualization, for which several solutions have been suggested in the graphics literature. We place streamlines according to the algorithm of Jobard and Lefer [1997]. The desired distance between adjacent streamlines is determined according to the tone of the underlying image. We then place another set of streamlines, this time using a vector field perpendicular to the original. These two sets of streamlines together form a grid. We bias the routing algorithm so that the walls that flow in the primary direction are considered later, and therefore have a lower probability of being removed during maze construction. Figure 4 illustrates the construction of a directional maze.



(a) (b) (c)

Figure 4: The construction of a directional maze. The vector field is defined in (a). In (b), black lines show the streamlines following the direction while blue lines show perpendicular streamlines. A final maze is given in (c).

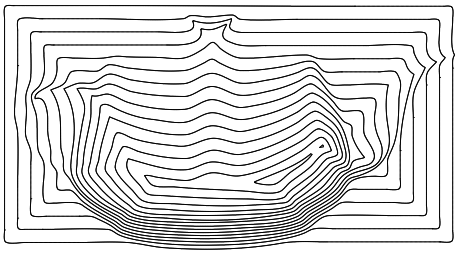


Figure 5: Example of concentric offset curves produced from a shaded image of a teapot.

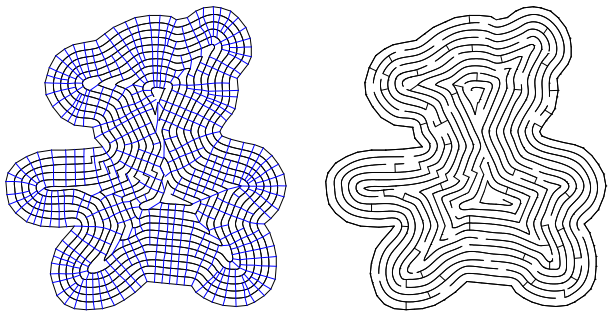
4.2 Spiral and vortex mazes

As Xu and Kaplan point out, spirals and vortices are powerful obfuscating devices in maze design [Xu and Kaplan 2007]. A vortex is a kind of multi-armed spiral in which multiple paths meet at a central junction. Their technique produces abstract, geometric mazes; we would like to support more flexible spiral regions that can participate in the stylized depiction of the source image.

While it may be possible to generate a spiral maze via a carefully constructed vector field, we have obtained better results by gradually shrinking the region boundary inwards, marking off concentric copies at regular intervals. We use an approach similar to Sethian's level set method [Sethian 1999]. We begin by placing seed points at the pixels that contain the region boundary. We then displace the seeds along image gradients towards the interior of the region. Each seed moves by an amount inversely proportional to local image tone. Periodically, we extract a contour from the current seed locations. This approach produces smoothly changing contours with controlled spacing, as illustrated in Figure 5.

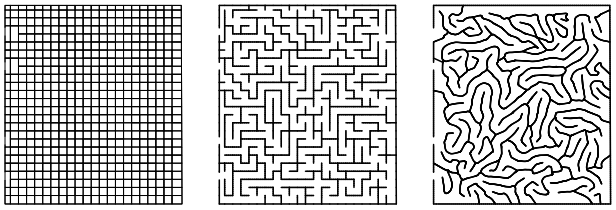
We then construct radial lines by sampling at regular intervals along the concentric contours. Each sample is connected to the closest sample on the next outer contour, resulting in a grid. We bias the routing algorithm so that radial walls are more likely to be broken than concentric walls. Figure 6 shows a spiral grid, and a maze constructed from it.

Our concentric contours are in fact compatible with the vortex construction technique of Xu and Kaplan. We make their technique



(a) (b)

Figure 6: The construction of a spiral maze. The underlying grid is shown in (b), and one possible final maze in (c).



(a) (b) (c)

Figure 7: The construction of a random maze. Horizontal and vertical lines form the grid in (a). A rectangular maze is constructed in (b) from the grid. After 200 iterations of the relaxation algorithm of Singh and Pedersen, we get the random maze pattern in (c).

available from our interface. It is occasionally useful, as it produces vortices where most paths get close to the region's centre.

4.3 Random mazes

There is no single correct way to define a "random" texture. We would certainly expect it to be isotropic and, in the absence of tone variation, evenly-spaced. Seeking inspiration in other work, we note that Berg [2001] uses a coral-like random texture for rough, natural shapes such as terrain and stones. This texture is reminiscent of reaction-diffusion patterns [Turk 1991], and reproduced by the geometric attraction-repulsion algorithm of Singh and Pedersen [2006].

Our random maze texture is very simple. We build a grid by taking the intersection of a regular square grid with the region. We then immediately build a maze using the algorithm of Section 5. Afterwards, we randomize the square maze in a post-process by passing it through Singh and Pedersen's algorithm. The result is a maze with the random appearance of their results, but with branching walls and maze structure. A simple example is presented in Figure 7.

Note that because it is a post-process, the relaxation algorithm can also be used to "loosen up" directional and spiral mazes to any desired degree. Relaxation can help suppress occasional artifacts in these other textures, such as passages that are too narrow.

4.4 User-defined lines

Some regions contain lines that should be drawn explicitly by the designer. These regions do not contribute heavily to the structure of the maze, but are important for depiction. We allow the user to sketch lines that are preserved as-is in the final maze. In the results in this paper, user-defined lines are used only for the face of the discus thrower in Figure 14(c).

5 User-specified solution paths

The collection of textures in the previous section can already be used to construct attractive mazes. Each texture fills a region with a grid. We can simply take the union of all these grids and use any standard maze algorithm to construct a random spanning tree of the combined cell graph.

In practice, we would like to offer the designer more explicit control over the routing of the solution path. The designer may wish to place start and end locations very close to one another, but prevent the system from linking them with a direct path. The solution path may also be linked semantically with the maze. In a portrait, for example, we may want the solution path to start in one eye and end in the other (without going straight across the bridge of the nose).

We allow the designer to sketch an arbitrary planar tree called the "solution tree" on top of the segmented maze footprint. The simplest such tree would be a single meandering path connecting start and end locations, but branches can be drawn to introduce predetermined dead ends. Examples of solution trees with and without branches appear in Figure 12(b) and Figure 13(b). In this section, we present an algorithm for building mazes that respect the layout of a given solution tree. Note that the paths of the maze will not necessarily follow the exact shape of the designer's sketch; doing so could lead to solutions that are too simple, or interfere with the ability to depict textures. The tree will be reflected in the routing of the paths within a region and the connectivity between the regions.

The restriction of the solution tree to a single region will consist of a planar forest (a set of non-intersecting planar trees). Some of the branches in the forest cross the region boundary. If we place a planar subdivision in the region, we can then identify the cells $S = \{s_1, \ldots, s_n\}$ through which the branches pass just before reaching the boundary. We open the boundary at these cells, yielding the inter-region connectivity suggested by the solution tree. We are left with the problem of breaking cell walls in the region's grid so that the resulting passages are connected according to the given planar forest.

Given a planar forest in a region and the cells S as given above, let us say that s_i and s_j are *co-routed* if they belong to the same tree. This equivalence relation partitions S into sets $\Delta_1, \ldots, \Delta_m$. We can now formalize the *maze routing problem* as follows: find a subgraph of the region's cell graph for which

(a) There is a passage from s_i to s_j if and only if they are co-routed;

(b) Every cell is connected to at least one s_i; and

(c) The subgraph contains no cycles.

A subgraph with these properties will yield a set of disjoint, intertwined sub-mazes that realize the correct routing within the region.

If we omit condition (b), and ask only that the s_i be connected correctly, we obtain what is known in computational geometry as the *vertex-disjoint subgraph problem*. It frequently appears in the literature as a generalization of the *vertex-disjoint path problem*, the special case in which $|\Delta_i| = 2$ for all i. Algorithms for the path problem typically solve the subgraph problem instead. Clearly, a solution to the disjoint subgraph problem can easily be extended into a solution to the maze routing problem by connecting any remaining cells to the subgraphs without introducing cycles.

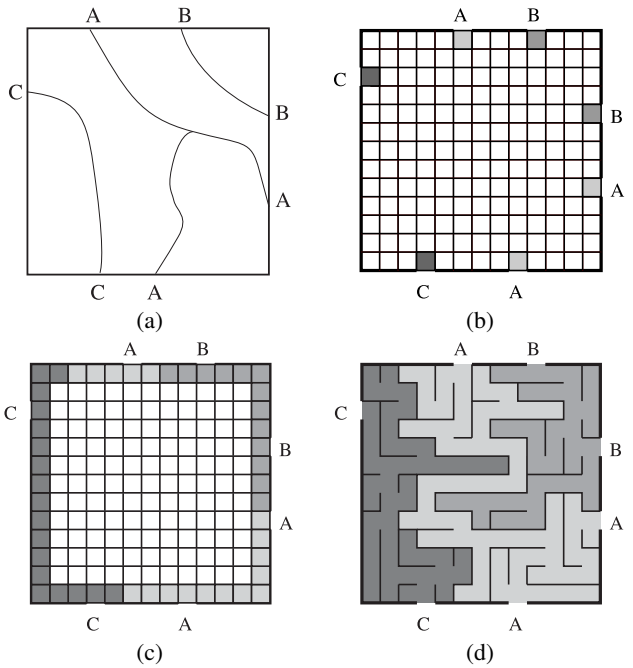


Figure 8: A demonstration of a maze with three non-interacting paths, constructed using the algorithm of Section 5. The restriction of the designer's sketched solution tree to the region is shown in (a), with points on the boundary labeled. In (b), the same labeling is applied to the cells in the grid that contain the locations where the trees meet the region boundary. The labeling is extended to the entire boundary in (c), and a finished maze is shown in (d).

In general, the vertex-disjoint subgraph problem is intractable, even for a planar graph [Karp 1975]. However, it can be solved efficiently in the special case that all the s_i lie on a single face of a planar graph [Robertson and Seymour 1986]. (In our case, the single face is the unbounded face corresponding to the region boundary.) In particular, Suzuki et al. demonstrate a linear-time algorithm that repeatedly connects vertices with paths along the outside of the planar graph and removes the path edges, leaving a simpler instance of the same problem [Suzuki et al. 1990].

Although the efficiency of the linear-time algorithm is attractive, it is unsuitable for constructing mazes. It is too greedy, in that it constructs direct paths that hug the region boundary. In contrast, we would like our maze passages to be as un-greedy as possible! We should avoid these short circuits, and force the solver to enter the depths of a region in order to traverse it. A greedy algorithm can also interfere with maze textures, since we might want to prefer breaking some walls over others.

We have developed our own algorithm that solves the maze routing problem. It is essentially a variation of Kruskal's algorithm that respects condition (a) above. An example is shown in Figure 8. In this example, the designer's sketched solution produces a forest of three trees in a region. We label the s_i with capital letters, where co-routed cells are given the same label.

We might imagine that it would suffice to track, for each cell, the label that "owns" it (if any). When considering whether to join two cells with a path, we could check not only whether doing so would introduce a cycle (as in Kruskal's algorithm), but also whether it would join cells owned by two different labels. This suggested approach is as efficient as Kruskal's algorithm, and it certainly prevents distinct labels from being connected. Unfortunately, as we can see in Figure 8(b), it fails even for simple cases. There is nothing preventing a C cell from being adjacent to a B cell, but any such adjacency would choke off the A cells, preventing them from being connected to each other.

Generally, two labels can never be permitted to get so close to one another that the other labels are unable to navigate simultaneously through the remaining gap. We can therefore extend Kruskal's algorithm by building a "minimum distance matrix", a symmetric integer matrix M with one row and column for each label. Entry $M_{\alpha\beta}$ records the minimum permitted path length between any α cell and any β cell in the grid. (Note that paths in this case may include consecutive cells that share only a vertex, and not necessarily an edge). Then, when deciding whether to break a wall, we iterate over every pair of labels α and β and decide whether breaking that wall would put an α cell too close to a β cell. We perform breadth-first searches from the cells on either side of the wall and compute d_α and d_β, the minimum distances to any α cell and β cell. If $d_\alpha + d_\beta < M_{\alpha\beta}$, it is illegal to break this wall.

We compute the minimum distance matrix by considering the circular sequence of labels around the region boundary. To compute $M_{\alpha\beta}$, we find all sub-sequences of labels that start at one of α or β and end at the other. Each sub-sequence defines some subset of the labels that can be found between α and β on the boundary. The intersection of these subsets tells us how many different labels must be given space to move between α and β. We set $M_{\alpha,\beta}$ to be one plus the size of this intersection.

There is one other complication with this algorithm. In Figure 8(b), an A cell might still choke off the C cells by connecting to the boundary between them. We avoid this possibility by propagating the labels on the s_i to all cells on the region boundary. We choose random division points between adjacent labels, which break the boundary cells into contiguous groups. As shown in Figure 8(c), we assign the cells in each group the label of the s_i in that group.

Our algorithm is roughly quadratic in the number of cells and the number of distinct labels. In practice, it solves the maze routing problem in a few seconds for regions containing thousands of cells. Note that in practice, a region rarely has a large number of different labels on its boundary.

Insofar as this approach is a variation on previous algorithms that find vertex-disjoint subgraphs on a disc, we require that all routing labels be placed on a region's boundary. In particular, this implies that our regions cannot contain holes if those holes are intersected by the solution path, as we would then have labels on more than one face in the cell graph. Robertson and Seymour [1986] show how to solve the case of a region with a single hole (the "disjoint subgraphs on a cylinder" problem). But in general, the complexity of the problem is exponential in the number of holes. We avoid creating holes in our regions, which we do not find to be a significant aesthetic constraint.

Although the algorithm is random, we have noticed that it still occasionally produces undesirable short circuits, as described above. We wish to avoid direct passages between consecutive s_i with the same label α. We can do so by assigning a random cell on the boundary between them a fresh label α'. The newly labeled cell will grow its own passages, disconnected from the rest of the maze. We pick a wall on the frontier of the passages and open it, transferring ownership of the passages to α. See Figure 9 for an example.

We can use a similar process to increase the global complexity of the maze. We place the fresh label at cells that share a section of the boundary between two regions. In one region we connect the label's passages to another label, as before. We then open the boundary

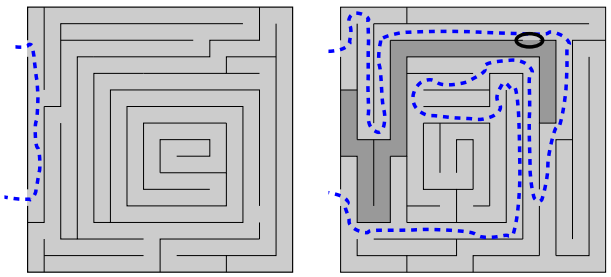


Figure 9: The elimination of a short circuit. The maze shown in (a) fills the square, but the path through the square is too short to make for a good puzzle. We introduce a fresh label between the two exits, which produces the darkly shaded passage in (b). We open the new passage at a random location (shown circled), creating a new maze with a longer solution.

where the labeled cells meet, linking in the passages in the other region. The result is a new branch that wanders off to a dead end in an adjacent region. In our system, we randomly introduce these extra branches. They make it more difficult to comprehend the structure of the maze via the openings between adjacent regions.

6 Additional effects

In order to further increase the range and visual appeal of our mazes, we have implemented two additional effects: tone reproduction and foreshortening.

6.1 Tone reproduction

Because we are building mazes from continuous-tone images, we would like to support some form of halftoning. Here we depart from Berg's clean line drawing style; he rarely uses more than one or two line weights in a maze, and is more concerned with suggesting form than tone. He does occasionally use multiple line weights in a single maze, usually to indicate distance.

The textures of Section 4 are defined in such a way that line density can be varied continuously across a region. Clearly, when placing lines we could simply map local image tone to a range of spacings. However, we must impose a minimal passage width in order to make the maze solvable in practice. We can also vary tone by adjusting line width, though again we should impose a minimum width so that lines are clearly visible.

Let G denote the lightness at a given point in a region. Imagine the simple case of covering the region with a set of horizontal lines. We can control S, the spacing between the centres of the lines, and W, their widths. The grey level approximated by given values of S and W is $(S-W)/S$. Therefore, to match our target lightness G, we obtain the relationship $W = S(1-G)$.

From the considerations above, we define a minimum line width $W_{\min}$ and minimum passage width $P_{\min}$. We also define $S_{\max}$, the largest acceptable line spacing. Immediately, we see that the darkest tone we can reproduce occurs when $S = S_{\max}$ and $S - W = P_{\min}$, giving lightness $G_{\min} = P_{\min}/S_{\max}$. Similarly, the lightest available tone is $G_{\max} = (S_{\max} - W_{\min})/S_{\max}$.

We can now define a mapping from image lightness G to values of S and W. Note that when both passage width and line width are minimized, the resulting lines have lightness $P_{\min}/(P_{\min}+W_{\min})$. We set a crossover value G_{thresh} to this lightness; for darker tones we will thicken the walls, and for lighter tones we will widen the passages. With a little algebra, we find that we should set $S = P_{\min}/G'$ and $W = P_{\min}(1-G')/G'$ when $G' \le G_{\text{thresh}}$, and $S = W_{\min}/(1-G')$ and $W = W_{\min}$ when $G' > G_{\text{thresh}}$. G' is computed by mapping G into the range $[G_{\min}, G_{\max}]$.

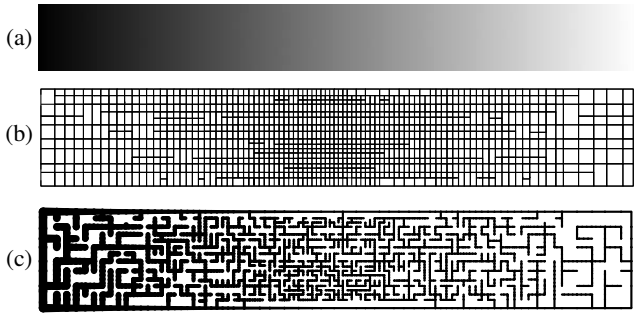


Figure 10: A demonstration of varying both line width and passage width to reproduce the continuous grey ramp of (a). The lines in (b) are drawn with fixed width to show their spacing. When drawn with varying thickness in (c), they capture the tone of the ramp.

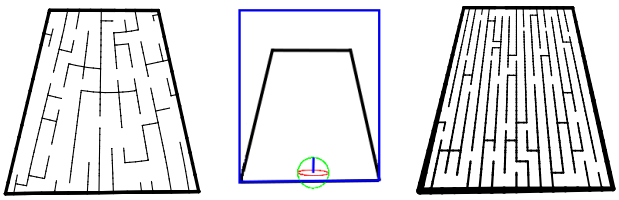


Figure 11: A foreshortened directional maze. We cannot simply create a directional maze with converging lines as in (a), because the perpendicular vector field does not have the correct perspective. In (b), the designer interactively rotates a normal estimator; the blue rectangle shows the result of undoing the perspective transformation implied by that normal. A maze is constructed and re-projected in (c).

As Figure 10 illustrates, this approach can approximate a wide range of tones while still permitting lines to delineate a maze.

6.2 Foreshortening

Foreshortening is a powerful artistic tool for depicting shape and perspective in line drawing. Nearly all of Berg's mazes use some form of foreshortening. Unfortunately, none of the textures defined in this paper can support foreshortening directly. As shown in Figure 11(a), a vector field can be used to capture converging lines in one direction, but the remaining lines are perpendicular to them in the image plane, not in the imagined 3D space.

We provide a simple tool that can add some foreshortening to maze regions. Inspired by the single view modeling system of Zhang et al. [2001], the designer can interactively place normal vectors on the image. The normal vectors are used to estimate a perspective transformation. We apply the inverse of this transformation to the region, build the maze, and map it back to the original region by re-applying the transformation. The interactive tool and the result are shown in Figure 11(b) and (c).

7 Implementation and results

Our system is implemented in C++, and outputs mazes in Postscript. We use the CGAL library [CGAL Editorial Board 2006] to store and manipulate regions and grids.

We have constructed a prototype user interface that lets the designer load an image, trace regions, assign a texture to each region, set parameters specific to the textures, sketch a solution tree, and choose start and end positions. Even for relatively complex images with up to two dozen regions, the design process requires only a few minutes of user interaction.

When the designer is satisfied with the settings, they can ask for a finished maze to be produced. Maze generation is not intended to be interactive. Still, final results are typically computed in a few minutes, permitting iterative refinement without too much waiting. We would like to experiment with a multithreaded approach, where each region's maze can be constructed in the background while the designer is working on the other regions.

Some sample results are shown in Figures 1, 12, 13 and 14.

8 Conclusions and Future Work

We have described a system for designing mazes that are stylized line drawings of images. Unlike previous work, we aim throughout to achieve a balance between considerations of complexity and aesthetics. The system handles the difficult task of placing individual lines while maintaining the constraints that make the final drawing a perfect maze. The designer is left with the high-level creative tasks of partitioning the image and deciding on the maze's appearance.

One aspect of Berg's mazes clearly missing in our work is his clear depiction of surface detail using the lines of the maze. His lines occasionally become faces or small decorative elements. To some extent, the user-defined lines of Section 4.4 can handle this case, though we would like to explore vision-based techniques for extracting salient curves from an image and placing them in the maze.

Berg's lines also convey higher-level repeated textures. The Assyrian Bull of Figure 2(b) exhibits textures on the wings and chest that cannot arise naturally through any of the textures of Section 4. Other mazes by Berg depict masonry, foliage and ornamentation in similar ways. We would like to extend our system to include semi-structured textures like these. An approach inspired by the "stroke textures" of Winkenbach and Salesin [1994] could apply here.

The perfect mazes we construct here are but one possible maze topology. It is also possible to construct mazes containing cycles, or indeed mazes with no dead ends at all ("Braid mazes", as Pullen calls them). These mazes can defeat the usual "keep your left hand on the wall" solving method by placing the start and end of the maze next to different connected components of walls. Human designers incorporate cycles freely, and they are a common (and frustrating) feature of hedge mazes. Of course, any perfect maze can be given cycles by breaking walls here and there, but it would be interesting to develop new maze generation algorithms that account for cycles, as well as guidelines for their effective use.

This paper has studied the problem of maze design from the perspective of non-photorealistic rendering. Having built our system as a proposed solution to the problem, we can now step back and ask about the quality of the puzzles it produces. Are they entertaining as mazes? We have enjoyed solving our examples, but we know of no concrete way to evaluate the effectiveness of a maze puzzle. We are fascinated by the question of how to measure the difficulty of a maze, which seems to depend on both its mathematical structure and on human psychology [Xu and Kaplan 2007]. Any progress on this question could help enhance our enjoyment of mazes and other puzzles on paper, at a human scale, and on the computer screen.

Acknowledgments

This work benefitted greatly from the valuable feedback and encouragement of George Hart, Allison Klein, Morgan McGuire, Karan Singh, and Michael Terry. Thanks also to Christopher Berg for permission to reproduce his maze, and to Mark R. Saunders, Leszek Scholz, Ronald Koster, and Claudio Pozas for permission to develop mazes based on their images. This research was supported by NSERC.

References

BERG, C. 2001. *Amazeing Art: Wonders of the Ancient World*. Harper Collins.

BERG, C., 2005. Amazeing art. `http://www.amazeingart.com`.

CGAL EDITORIAL BOARD. 2006. *CGAL-3.2 User and Reference Manual*. `http://www.cgal.org`.

CONCEPTIS LIMITED, 2006. Conceptis puzzles. `http://www.conceptispuzzles.com`.

CORMEN, T. H., LEIESERSON, C. E., AND RIVEST, R. L. 1992. *Introduction to Algorithms*. MIT Press.

FISHER, A. 2006. *The Amazing Book of Mazes*. Harry N. Abrams, Inc.

HAYS, J., AND ESSA, I. 2004. Image and video based painterly animation. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM Press, 113–120.

HUANG, J., AND FEIGENSON, G. W. 1999. A microscopic interaction model of maximum solubility of cholesterol in lipid bilayers. *Biophysical Journal 76*, 4, 2142–2157.

JOBARD, B., AND LEFER, W. 1997. Creating evenly-spaced streamlines of arbitrary density. In *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop in Boulogne-sur-Mer, France*, Springer Verlag, 43–56.

KAPLAN, C. S., AND BOSCH, R. 2005. TSP art. In *Bridges 2005: Mathematical Connections in Art, Music and Science*, 301–308.

KARP, R. M. 1975. On the computational complexity of combinatorial problems. *Networks*, 5, 45–68.

KERN, H. 2000. *Through the Labyrinth: designs and meanings over 5000 years*. Prestel.

MORALES, J. E., 2006. Virtual Mo. `http://www.virtualmo.com`.

MORTENSEN, E. N., AND BARRETT, W. A. 1995. Intelligent scissors for image composition. ACM Press, 191–198.

OSTROMOUKHOV, V. 1999. Digital facial engraving. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 417–424.

PEATFIELD, G., 2005. Maze creator. `http://www.mazecreator.com/`.

PEDERSEN, H., AND SINGH, K. 2006. Organic labyrinths and mazes. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM Press, 79–86.

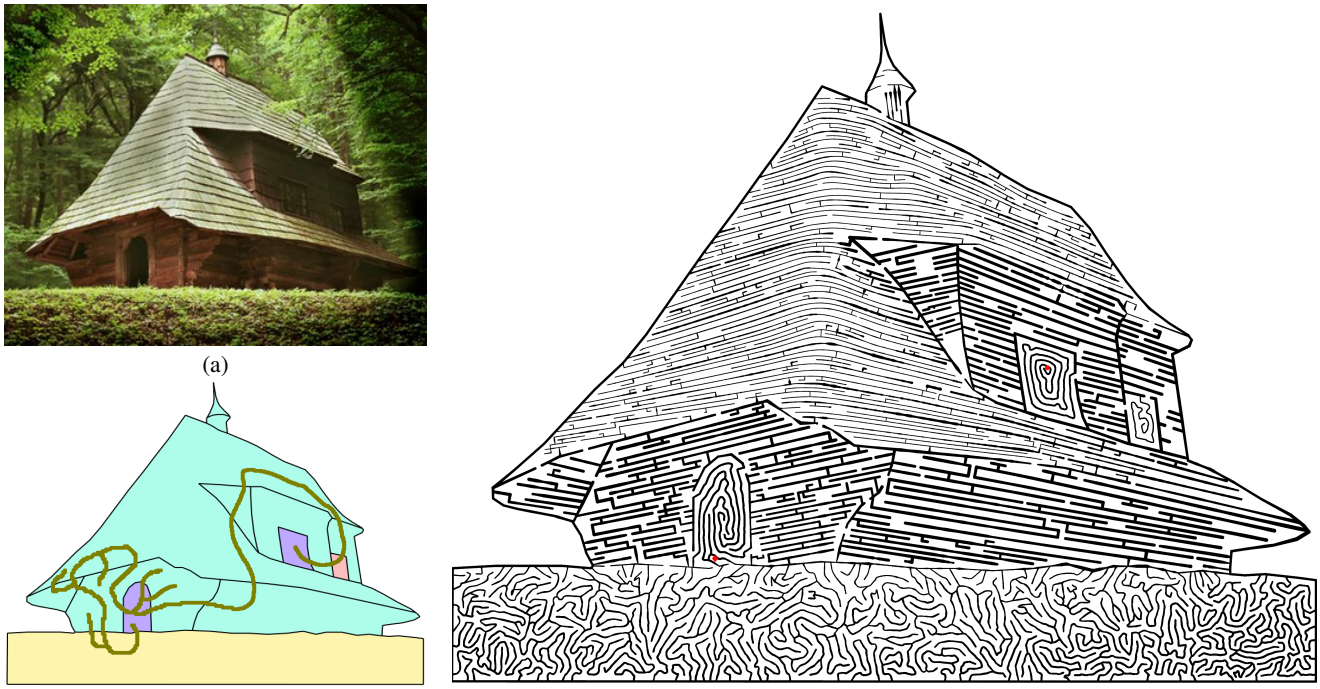


Figure 12: A sample maze. The original photograph is shown in (a). In (b), the designer has partitioned the maze footprint into regions and sketched a solution tree. Photograph by Leszek Scholz.

PULLEN, W. D., 2005. Think labyrinth. `http://www.astrolog.org/labyrnth/maze.htm`.

ROBERTSON, N., AND SEYMOUR, P. D. 1986. Graph minors. VI. disjoint paths across a disc. *J. Comb. Theory Ser. B 41*, 1, 115–138.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 401–406.

SETHIAN, J. A. 1999. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press.

SHIVERS, O., 2005. Maze generation. `http://www.cc.gatech.edu/~shivers/mazes.html`.

STEVENS, P. S. 1974. *Patterns in Nature*. Little, Brown.

SUZUKI, H., AKAMA, T., AND NISHIZEKI, T. 1990. Finding steiner forests in planar graphs. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 444–453.

TURK, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM Press, 289–298.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, 91–100.

XU, J., AND KAPLAN, C. S. 2007. Vortex maze construction. *Journal of Mathematics and the Arts 1*, 1 (March), 7–20.

ZHANG, L., DUGAS-PHOCION, G., SAMSON, J., AND SEITZ, S. 2001. Single view modeling of free-form scenes. In *Proc. of CVPR, 2001*, vol. 1, 990–997.

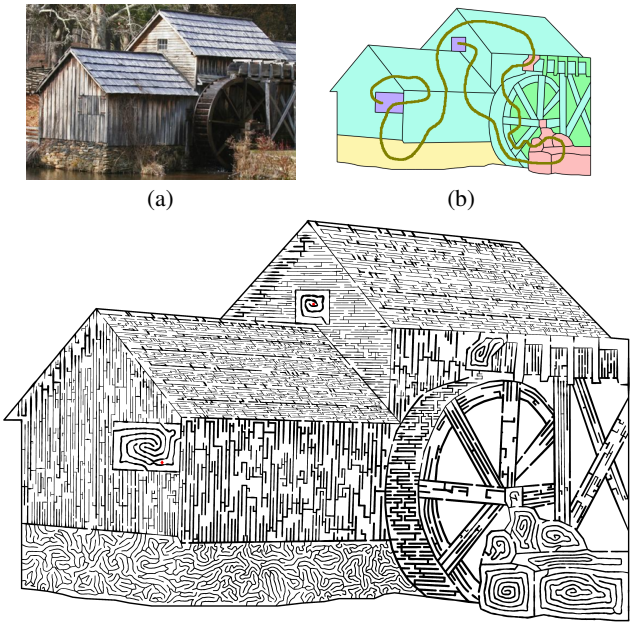
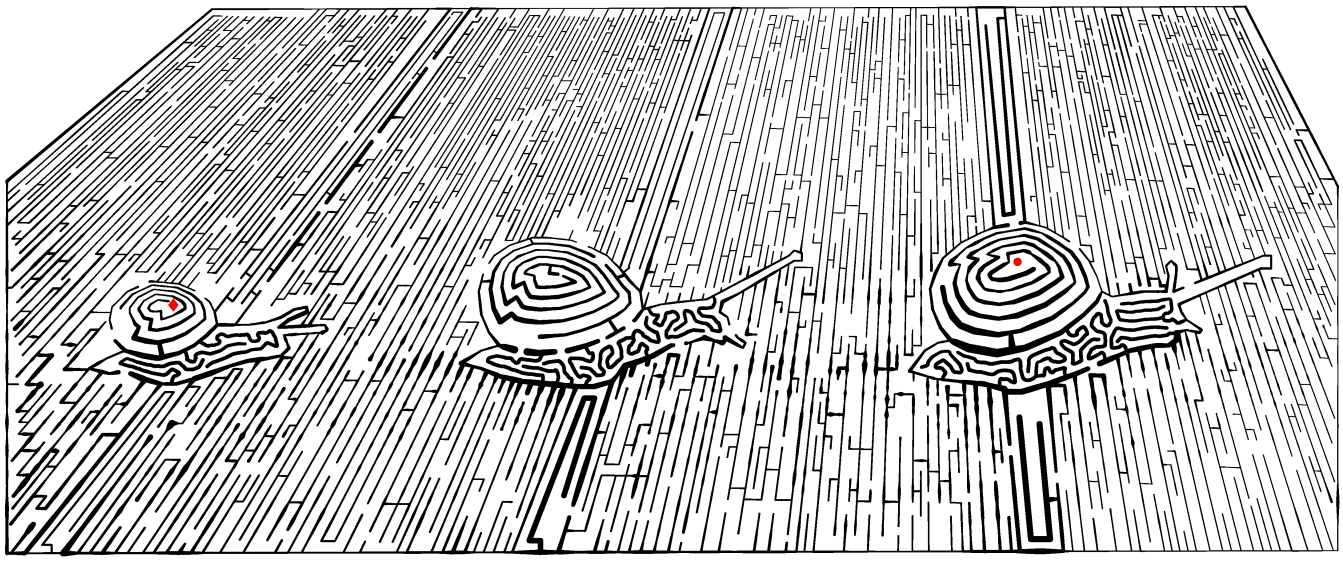
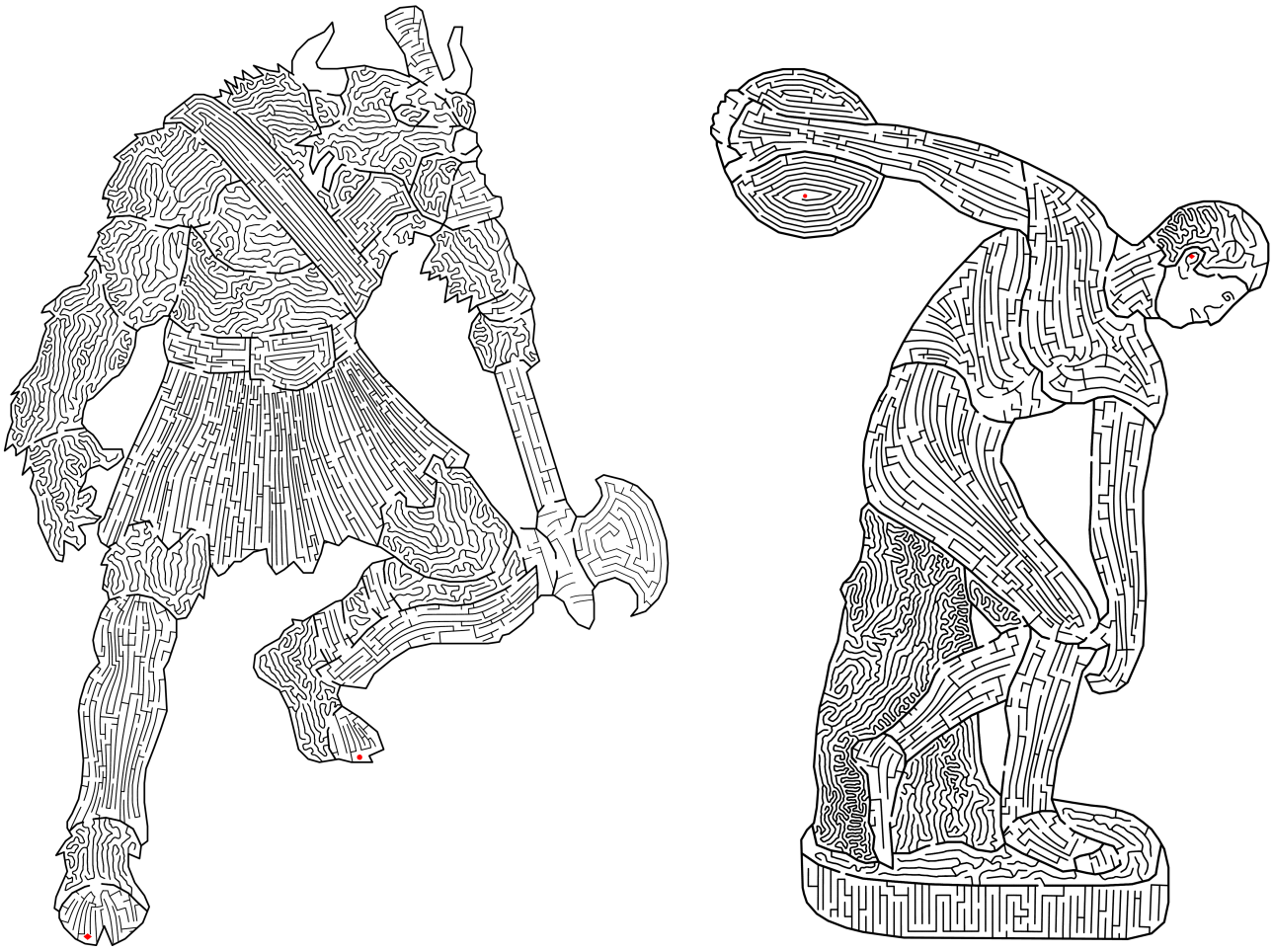


Figure 13: A sample maze based on a photograph of a mill. Original photograph by Mark R. Saunders.



(a)



(b) (c)

Figure 14: More examples that use the techniques presented in this paper: Snails (a), The Minotaur (b), and Myron's Discus Thrower (c). Each of these examples was produced after about ten minutes of user interaction and five minutes of computation. Original images for (a) and (b) by Ronald Koster and Claudio Pozas, respectively.