Maze Generation from Video

MARIE PIZZINI, SERVANE DESVIGNES, CELIO BOULAY, Télécom Paris, France

.

Abstract: Mazes are impressive and fascinating pieces of art. Previous works have shown that it is possible to accurately create a maze on top of an image while keeping the main features of the image in the features of the maze. Until now, only one research paper existed on dynamic maze generation. Our objective was to continue their research and try to improve the model. Our work focused on implementing a great maze generating model, and improving the coherence between successive frames from a video. Using several methods explained in this paper, we were able to offer our own algorithm to generate mazes sequences from videos.
Source code is available here: https://github.com/Surimi91/IGR205

1 INTRODUCTION

Maze generation is an area of computational geometry and algorithm design, with applications ranging from game development to procedural content generation. Traditionally, mazes have been generated using various algorithms that create intricate pathways and dead ends, challenging the solver to find the correct route from the start to the finish. The typical resources we could find on maze generation focus on static images or predefined patterns. However, an unexplored side of this field involves dynamic content, such as videos. Unlike static images, videos provide a sequence of frames that capture motion and changes over time, which can be leveraged to create more complex mazes. Despite the potential, there has been limited research on this topic (only one academic paper, which was not published). Maze generation from images usually involves algorithms like Reaction-Diffusion (RD) [1] to get patterns, A* or Dijkstra algorithm to turn it into a solvable maze [6]. But when it comes to videos, the process becomes more complex. Videos consist of a series of frames, each representing a static image at a different moment in time. The temporal dimension adds a layer of complexity, as the maze generation algorithm must now account for motion, changes in the scene, and possibly varying levels of detail across frames. This dynamic nature requires advanced techniques in image processing, such as optical flow analysis [x] or frame differencing to extract meaningful patterns that can be translated into maze structures. The paper we found [2] on this subject explores the use of video data to create mazes by analyzing motion and changes within the frames. The approach involves detecting edges and features in the video frames, tracking their movement over time, and then using this information to construct a maze that evolves with the video content. We could imagine using such generation processes in advanced video game scenarios where the environment adapts dynamically to the player's actions.

2 LITERATURE REVIEW

Pedersen and Singh [3] created a model to transform simple shapes into organic labyrinths. However, this method often fails to preserve complex interior structures without detailed manual segmentation. Xu and Kaplan [4] developed procedural algorithms to generate maze templates. Their approach also requires careful image segmentation to maintain salient structures, which can be labor-intensive for complex images. Reaction–diffusion systems, here modeled by the Gray-Scott model, describe changes in the concentration of chemical substances over space and time through local reactions and diffusion. Alan Turing theorized that these systems cause patterns seen in zebras, snakes, or corals. In "Evolving Maze form Images", Wan, Liu, Wong, and Leung [1] introduce a novel RD simulator developed within the framework of cellular neural networks (CNN). Unlike traditional RD models, this simulator can evolve stripe patterns that resemble the source images, preserving the salient interior structures of the source images during the evolution process. This method adapts to the underlying image, making it ideal for creating mazes that change over time and interact with user inputs. In "Moving Mazes", Bommadevara and Flahaut extend RD simulation to moving images, ensuring frame coherence and smooth transitions by mapping corresponding walls between frames using geometric mean calculations. Their paper addresses the computational complexity of real-time maze creation for high-resolution videos and the challenge of balancing maze complexity with image fidelity.

Flood-filling [5] is commonly used in image processing to detect and fill connected regions, and proves instrumental in the context of maze generation by enabling the identification of chambers within the random patterns generated. This technique is particularly valuable when working with video frames, as it aids in segmenting the maze into manageable sections, ensuring that each part of the maze remains coherent and solvable across successive frames. Graph solving algorithms [6], [ ], [ ] are essential for ensuring the solvability of mazes. When applied to maze generation, such algorithms guarantee that there is a viable path from the start to the finish, regardless of the initial structure of the maze.

3 TECHNICAL DETAILS

3.1 Video processing

For our research we essentially used gif, as they are easier to split into frames. Videos would require an extra step – making sure the frame is clear enough to be processed – that we didn't have time to tackle. Our model processes square gif for simplicity but can be adapted to any video format.

3.2 Reaction-Diffusion process

First step in converting the image to a grayscale pixel grid. Those value also need a remap from [0,255] to [0,1] to [r1,r2] included in [0,1]. r1 (0.396) and r2 (0.588) are given by [1]. We'll refer to this grid as X(t=0), the first state of the simulation. The model evolves according to the following equations:

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

$$A = \begin{bmatrix} -0.25 & -1.0 & -1.5 & -1.0 & -0.25 \\ -1.0 & 2.5 & 7.0 & 2.5 & -1.0 \\ -1.5 & 7.0 & -23.5 & 7.0 & -1.5 \\ -1.0 & 2.5 & 7.0 & 2.5 & -1.0 \\ -0.25 & -1.0 & -1.5 & -1.0 & -0.25 \end{bmatrix} \tag{2}$$

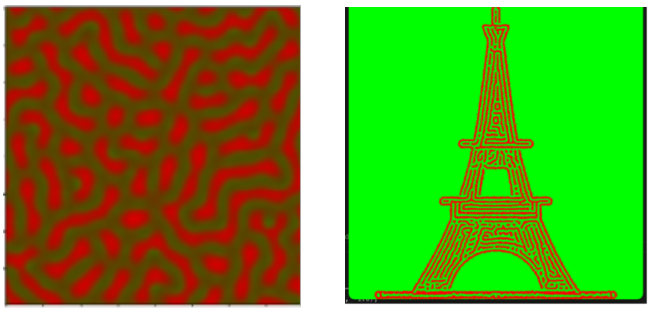

Fig. 1. RD-Models comparison

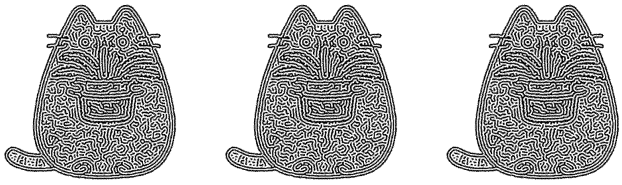

Fig. 2. Examples of patterns generated

With A the 5x5 kernel matrix. X(t) the state of the grid at step t, Y the response of the model to the X state, and I a constant input. Y is calculated as follows:

$$Y_{i,j}^{(t)} = \frac{1}{2}\left[\left|X_{i,j}^{(t)} + 1\right| - \left|X_{i,j}^{(t)} - 1\right|\right] \tag{3}$$

We can solve this system:

$$X_{i,j}^{(t+1)} = X_{i,j}^{(t)} + \sum_{k,l\in\eta(i,j)} a_{k-i,l-j} Y_{k,l}^{(t)} + I_{i,j} \tag{4}$$

A basic reaction diffusion model generates random patterns. Here, in order to make it look similar to the image, we take X(t=0) as the image converted to grayscale. We also make sure that the image is fed into the system at each step. This is done by setting I proportional to X(t=0) [1]

$$X^{(0)} = c_1 P_m, \quad I = c_2 P_m \tag{5}$$

This model allows the generation of patterns.

The next two steps are crucial in order to get a solvable maze. Indeed, the pattern we generated looks like a maze, but has no path, nor beginning/end.

3.3 Flood-filling and chamber detection

We define as chambers every white space enclosed by walls. In order to identify the chambers on our pattern, we use a flood-filling algorithm [5]. The flood fill process begins by adding an initial point to a stack, which serves as the queue for points to be processed. During the main loop, the algorithm continues to process points from the stack until it's empty. Each point is first checked to ensure it is within the grid boundaries and hasn't been previously marked or doesn't match the target value. If valid, the point is marked with a specified marker and its neighboring points (left, right, up, down) are added to the stack for further processing. This method is particularly effective for detecting chambers or distinct regions within a grid or image, as it allows the identification and segmentation of contiguous areas sharing similar properties. The process concludes when the stack is empty, returning a list of all points marked during the operation, effectively mapping out the detected chambers.

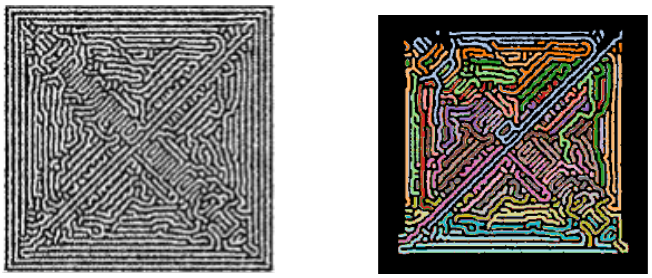

Fig. 3. Chamber detection

3.4 Dijkstra algorithm and maze processing

Once we have localized every chamber, we represent them on a graph based on their coordinates (Fig. 7. a). The start/end points are chosen randomly, the only condition being that they need to be on the extremities of the figure and as far away as possible. Dijkstra algorithm works by iteratively selecting the node with the smallest known distance from the start node, then exploring its neighboring nodes to update their shortest distances. Each time a node is visited, the shortest path to that node is potentially updated if a shorter path is found via the current node. This process repeats until we reach the end node, or all possible paths are exhausted. The result is the shortest path through the graph, navigating efficiently through the maze from start to finish based on the weighted edges that may represent distance or difficulty of passage between chambers. Every edge here weights the same, thus giving us the shortest path. In order to produce more randomness and get longer paths, we could adjust the weight of some edges. Then we break some parts of the walls between two consecutive chambers in the path returned by Dijkstra. This created one unique chamber: the path from start to finish (Fig. 7. b). At the point, we only created one path, and a quite obvious one, looking at the result (Fig. 7. b). In order to "hide" the main path, we need to create many others. To do so, we can break random walls, or created other paths (with Dijkstra) connecting completely random chambers.

3.5 Remapping and frame coherence

Cette sous-partie c'est un peu le coeur du projet donc prévoir presque une page.

While it is easy to generate a maze on each frame a of gif, it is way more challenging to make sure those mazes are similar. Indeed, the generation looks like the image – in the way it follows the salient structures – but its interior is still randomly generated. Fig. 8. shows the differences between patterns generated on successive

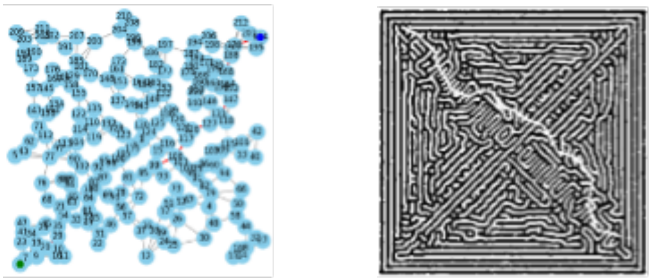

Fig. 4. Changer la qualité des images et bien les organiser

frames. The contours stay roughly the same, but everything inside is completely different. While this still offers great visuals, we wanted to develop coherence between frames.

4 MODIFICATIONS & IMPROVEMENTS

Time comparisons

Differences with [2]

-Pipeline and multithreading (6th part of technical details) Efficiency

It was important for us to get an optimized process. We chose to use C++ for the Reaction-Diffusion step as it requires a great number of calculations. (Give the numbers) Python for the global pipeline, easier to use and is efficient to centralize the data.

5 OBSERVATIONS & FUTURE WORKS

When studying the case of frame coherence, our fellow IPP predecessors [2] thought about using a CNN for the remapping. We attempted that by training a model.

From an innovation point of view, the first step was generating on images, then on a series of image. We could now only expect it to expand on 3D environments. [3] and [10] explain how we could develop a RD-simulator for 3D structures.

6 CONCLUSION

In conclusion, our research extended maze generation from static images to dynamic videos. Using techniques like reaction-diffusion and optical flow analysis, we developed an algorithm that creates intricate maze sequences from video frames while maintaining visual coherence.

Building on previous studies, we addressed challenges unique to video data, ensuring solvable and visually appealing mazes through flood-filling for chamber detection and the Dijkstra algorithm for pathfinding. Our optimized process, using C++ and Python, demonstrates the feasibility of short-time maze generation.

Future work could explore 3D environments and more sophisticated pathfinding algorithms, further enhancing our model's applicability. + speed Overall, we tried our best to offer a robust framework for dynamic, video-based mazes.

REFERENCES

[1] Wan L, Liu X, Wong TT, Leung CS. "Evolving mazes from images". IEEE Trans Vis Comput Graph. 2010 Mar-Apr;16(2):287-97. doi: 10.1109/TVCG.2009.85. PMID: 20075488.

[2] A. Bommadevara and A. Flahaut "Moving Mazes"

[3] Pedersen, Hans Køhling and Karan Singh. "Organic labyrinths and mazes." International Symposium on Non-Photorealistic Animation and Rendering (2006).

[4] Jie Xu and Craig S. Kaplan. 2007. Image-guided maze construction. ACM Trans. Graph. 26, 3 (July 2007), 29–es. https://doi.org/10.1145/1276377.1276414

[5] Burtsev, S. V. and Ye.P. Kuzmin. "An efficient flood-filling algorithm." Comput. Graph. 17 (1993): 549-561.

[6] R. Kumar, K. Sharath. "Path Finding Visualisation in Mazes Using Various Algorithms" DOI: 10.17148/IARJSET.2022.96130

[7]

[8]

[9]

[10] Turk, Greg. "Generating textures on arbitrary surfaces using reaction-diffusion." Acm Siggraph Computer Graphics 25.4 (1991): 289-298

[11]