

3D Model Generation from Freehand Drawings

Pedro Rossa*

Daniel Camozzato

Fernando Marson

Rafael Hocevar

Universidade do Vale do Rio dos Sinos, Brasil

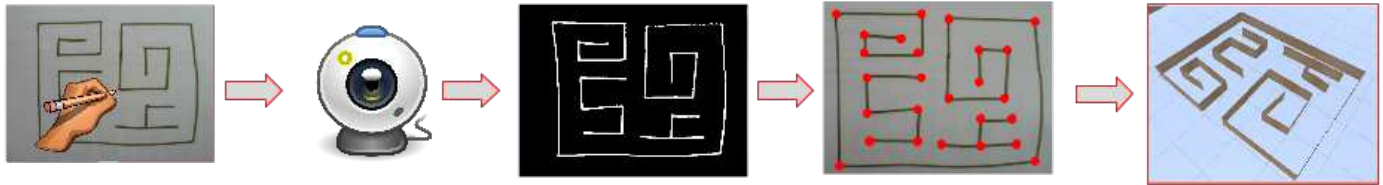


Figura 1: a) hand-drawn image; b) webcam analysis; c) binary Image; d) detected corners; e) virtual model representation.

ABSTRACT

We propose a model to analyze freehand labyrinth drawings in order to generate a 3D model of it. The drawing is captured using a webcam and the images are then analyzed by a computer vision algorithm that detects the labyrinth's 2D lines and builds the walls by extrusion. We aim to use the drawing itself as a marker, overlaying the 3D model to create an augmented reality effect.

Keywords: Computer Vision, Augmented Reality.

1 INTRODUCTION

Computer vision applications have become popular in different areas of knowledge in recent years. Augmented reality is an area that depends heavily on advances in computer vision techniques, and is necessary for the development of numerous applications for entertainment, education and business. One of the main challenges when working with computer vision is the need for controlled environments to enable image capture of acceptable quality.

In this article, we present a model to recognize hand-drawn labyrinth designs from images captured with a webcam. We perform a luminosity calibration applying filters to detect the labyrinth's edges. With this information, we then perform a sequence of steps to connect each pair of corners forming a line, resulting in points which can be used to create a 3D model through an extrusion process.

2 RELATED WORK

Several methods have been presented to create 3D building models from CAD files and scanned images [6], [7], [8]. While CAD-based systems avoid the overhead of image processing, many drawings are still done on paper and saved as scanned images. According to Yin et al. [8], these image-based systems are not fully automated, and vectorization and symbol recognition remain open issues.

Heras et al. [3] present a method in which a set of assumptions about the graphic representation of walls is used to generate segmentation candidates and select the one which better characterizes the walls in a given floor plan. This allows the method to identify walls in floor plans drawn using different graphic styles.

*e-mail: prossar@unisinos.br

Ahmed et al. [1] present an approach using successive morphological operations to recover walls represented with lines of different thicknesses. Macé et al. [5] present a technique which uses computer vision algorithms to extract lines from the input image, and which then identifies walls among the lines by using the lines as markers upon the input image. Walls are detected considering the directions of the lines and the texture between each two lines in the input image.

These image-based systems are suitable for use in recovering floor plans drawn using specific graphic styles for walls and openings. However, the reviewed methods are not fully automated, and vectorization, symbol recognition and robust image recognition in uncontrolled environments remain open issues. In our case the input is a hand-drawn sketch captured with a webcam in an uncontrolled environment. Therefore, we focus on automation and robustness.

3 THE MODEL

The first step is to draw the labyrinth layout on a clean white sheet of paper using horizontal and vertical lines only, as shown in Figure 1a. It is preferred to use a thick-tipped pen of dark color to achieve better results.

To generate a 3D model from a freehand drawing, we propose a sequence of steps. First, the camera is calibrated to reduce artifacts caused by performing image capture in an uncontrolled environment. Then, the full color input image is filtered, generating a two-color bitmap, with the background in black and the foreground (the labyrinth walls) in white. This is a preparatory step, and the detected edges must be further processed to create 2D vertices and lines which can be extruded to create a 3D model. Next, a corner detection algorithm is used to identify intersections between edges, such as corners in the labyrinth. Finally, the detected corners are used as markers to scan the input image, creating 2D lines upon the labyrinth drawn by the user. These steps are further detailed in the following sections.

3.1 Brightness Calibration

Brightness may vary greatly according to the environment in which the camera is operating. In order to solve this problem and achieve better results, the user is asked to do a quick calibration, which will consider the ambient brightness when performing the image capture. To perform the calibration, a clean sheet with the same

color used for the drawing is placed in front of the camera for a few seconds to collect some frames.

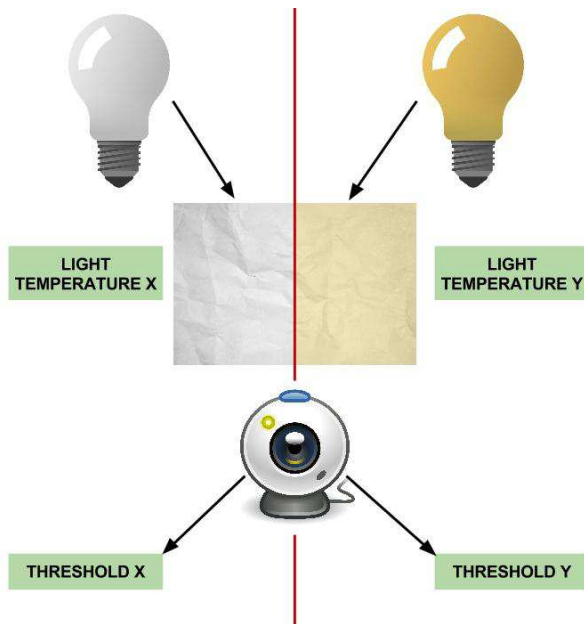


Figure 2: Representative image of the white balance algorithm.

This step is important because it enables the algorithm to adapt according to the ambient luminosity, which allows the detection to be more precise as seen in the Figure 2.

3.2 Color Filtering

The full color input image is filtered, creating a two-color bitmap. In this filtered image, the background (empty space) is black and the foreground (labyrinth wall) is white. This simplified image is further processed in a corner detection step, and also used later in a line scanning step. Our model detects both vertical and horizontal lines with a certain degree of freedom, such that lines do not need to be perfectly aligned (see Figure 1a), i.e., small changes in direction are detected and considered.

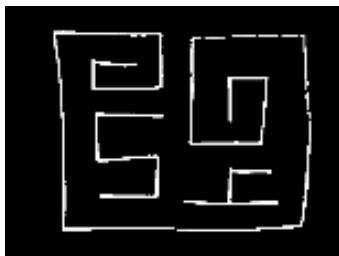


Figure 3: Two color image to analyze corner points.

3.3 Detection of Corners

After detecting the edges in the drawing, a second step is performed to detect corners and intersections between edges. Corners and intersections are features in the image, marked by a change in the edge's direction and, as a result, the gradient of the image has a high variation. This variation can be detected using the Harris filter [4], which sweeps a region of interest over the image, identifying intensity variations.

The Harris filter [4] generates multiple corner candidates for each actual corner between two lines, making it necessary to eliminate duplicates. The x and y coordinates for each corner candidate are stored in a buffer, and a neighborhood analysis is performed to eliminate duplicated corners. The neighborhood analysis eliminates each corner candidate beyond the first within a user-defined threshold. Iterating over each corner candidate, the distance to all other corners is calculated, and any corners closer than the threshold are removed.

The result is a set of coordinates for corners between the lines in the image (as shown in Figure 1d). The Harris filter may occasionally create a false positive, such as identifying a corner in the middle of a straight line. However, this does not cause a problem as the extra corner will simply be an extra vertex in our model.

3.4 Line Scanning

The corners detected in the previous step can be used as vertices for the 3D model extrusion. However, we must first identify which pairs of corners are actually connected with lines. Thus, a scanning process is performed as follows.

First, the previously detected corners are used as markers in the input image. Then, for each corner, we analyze the nearest lines in the original drawing by sweeping horizontally and vertically with a search window. The process of line detection occurs by following a set of rules defined as follows: the group of points detected by the Harris filter, and each of its values is analyzed horizontally and vertically. The diagram presented in Figure 4 demonstrate this concept.

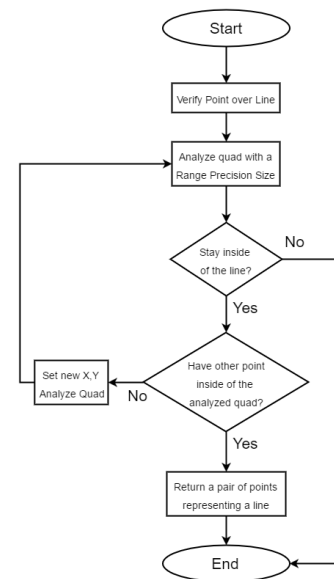


Figure 4: Corner analysis fluxogram.

Thus, there are three possibilities:

- First, a corner may be found before the line ends, representing a junction of lines (Figure 5.a);
- Second, a corner may be found at the end of a scan, representing a full line (Figure 5.b);
- Third, a corner may be found off the line, representing an error in the corner detection. This coordinate is then removed from the list (Figure 5.c).

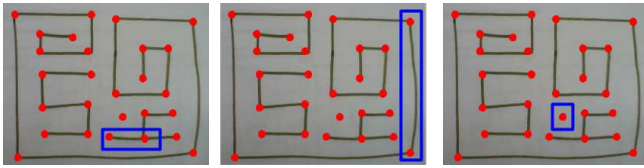


Figura 5: A hand-drawn design with the detected corners represented by red dots. From left to right: a corner detected in a junction of lines, a full line, and an isolated corner, which is not upon a line.

3.5 3D Model Creation

Using the pairs of corners obtained in the previous module (Section 3.4), a floor plan is defined for the 3D representation of the drawing, as shown in Figure 6. An example of this file can be seen in Listing ?? . Each line of the file contains two points, with each coordinate X and Y - respectively, separated by a '#' symbol. So, each line of the file represents a connection between two corners.

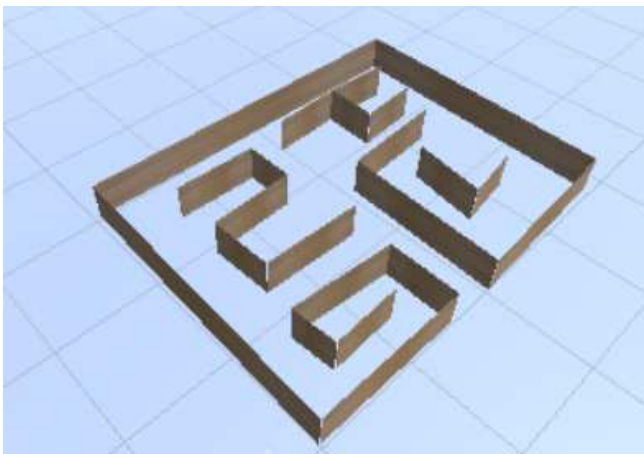


Figura 6: A 3D model result loaded in a graphics engine.

```
56#200#76#250
114#340#120#398
138#130#157#132
...|
```

Figura 7: An example of text file containing the initial three pair of corners used to build the 3D model.

This file can be loaded by a 3D graphics engine to build the labyrinth walls. Each wall is extruded from the pairs of points contained in the file and connected to each other to created the corners.

The next section presents some results obtained through a prototype implemented to test our model.

4 RESULTS

In order to test our computational model, we developed a prototype using the OpenCV library [2] and the Unity 3D ¹. The experiments

¹ www.unity3d.com

were performed using an Intel i7 processor equipped with NVidia GeForce 830M GPU and a Logitech HD WebCam C270. To test the robustness of the algorithm, we performed image capture in different environments, with different light sources. The results of our corner detection were satisfactory, with full corner detection both under natural light (sun and cloudy weather) and under artificial lights, such as incandescent and fluorescent lamps. Environments with fluorescent lamps can present light flickering. This issue is dealt with by the use of the image buffer (see Section 3.1), which allows the flickering of illumination to be ignored. Moreover, even in low light (with shadows over the design), the detection is still acceptable. The occurrence of isolated points increases but they are automatically eliminated. The three-dimensional representation of the design was performed for different types of designs, and we observe a limitation where lines must not be placed too closely, to avoid overlapping during corner detection.

Figure 8 and Figure 9 presents results obtained from two different drawing as input to our prototype.

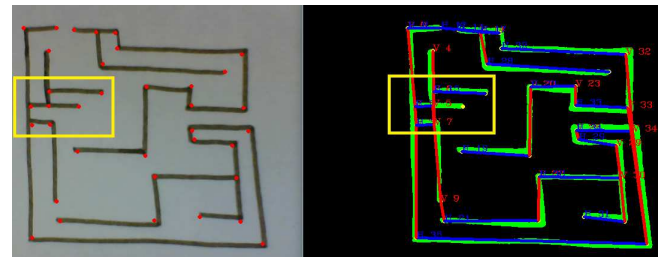


Figura 8: On the left, the original drawing with detected corners indicated by red dots. On the right, the detection of edges formed by the pairs of corners. Inside of the yellow rectangle it is possible to see spots that are very close, generating an error on line detection.

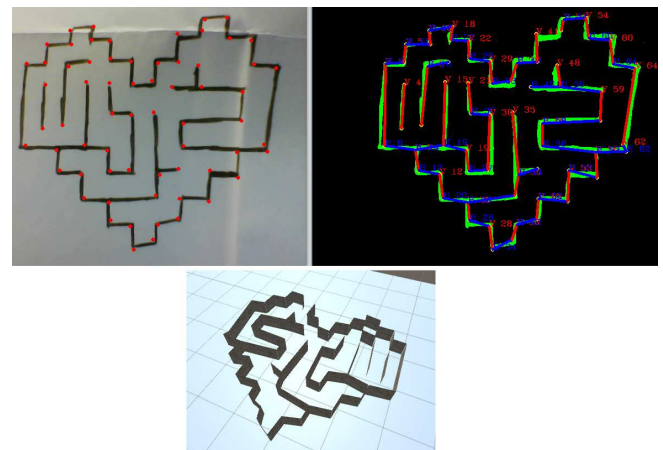


Figura 9: Another example of labyrinth. On the upper left, the original drawing with detected corners indicated by red dots. On the upper right, the detection of edges formed by the pairs of corners, and below is the corresponding 3D model.

5 FINAL CONSIDERATIONS

In this paper, we presented a model to generate a 3D representation from a freehand drawing designed by the user. The model receives as input the 2D drawing of a labyrinth composed by horizontal and vertical lines captured by a camera, and the final output is a 3D model extruded from the labyrinth's floor plan, initially designed by

the user. In order to deal with light variations, a brightness calibration is performed before the detection process begins, allowing the algorithm to achieve more accurate results. After calibration, the model follows a sequence of steps, in which each detection module output serves as input to the next one, as seen in Figure 1. First, the algorithm detects each line in the original drawing. In the second step, line intersections are perceived, i.e., the corners of the labyrinth. Next, we search for connections between these corner points by analyzing the line extrapolations around these points in the original drawing. The final step takes into account all these connections, which represent the labyrinth's floor plan, to generate the walls for a 3D model. This model is then exported to a file that can be loaded by any 3D engine, as explained in Section 3.5.

In order to test our model, we developed a prototype in C++ programming language and the OpenCV library [2]. We tested the model's robustness by using several labyrinth drawings as input and capturing these images in different light conditions. We believe that the results were satisfactory, since the program was able to create the floor plans with good precision as demonstrated in Section 4 and generate the 3D model representation for graphic engines.

6 FUTURE WORK

Our model can be employed in a wide variety of applications. For instance, an architect can use this model to facilitate the generation of a 3D model by drawing an apartment layout. In future versions, the model could be extended to automatically detect doors, windows and furniture by analyzing other drawing patterns.

Although the model can be applied in other areas, our efforts will be focused on creating virtual reality games. Therefore, as future work, we have defined three distinct objectives that will improve the model and allow new features:

- First, the detection of diagonal lines and curves, allowing more complex designs to be created;
- Second, the application of augmented reality, using the design itself as a marker to allow the resulting 3D model to be viewed upon the original drawing using a mobile phone;
- Finally, the use of the 3D environment together with a head mounted display, enabling the user to navigate inside the labyrinth generated from the freehand design created by the user.

REFERENCES

- [1] S. Ahmed, M. Liwicki, M. Weber, and A. Dengel. Automatic room detection and room labeling from architectural floor plans. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 339–343. IEEE, 2012.
- [2] G. Bradski. The OpenCV Library. *Dr. Dobbs' Journal of Software Tools*, 2000.
- [3] L.-P. de las Heras, D. Fernandez, E. Valveny, J. Lladós, and G. Sanchez. Unsupervised wall detector in architectural floor plans. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1245–1249. IEEE, 2013.
- [4] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.
- [5] S. Macé, H. Locteau, E. Valveny, and S. Tabbone. A system to detect rooms in architectural floor plan images. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 167–174. ACM, 2010.
- [6] C. So, G. Baciú, and H. Sun. Reconstruction of 3d virtual buildings from 2d architectural floor plans. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 17–23. ACM, 1998.
- [7] W. Y. Yeung. Creation of 3d model from 2d floor plan. 2008.
- [8] X. Yin, P. Wonka, and A. Razdan. Generating 3d building models from architectural drawings: A survey. *IEEE Computer Graphics and Applications*, (1):20–30, 2009.