Algorithmic improvements to an exact region-filling technique

Antonio Elias Fabris[1], **Valério Ramos Batista**[2]
[1] *IME-USP, r. do Matão 1010, 05508-090 São Paulo-SP, Brazil*
[2] *CMCC-UFABC, av. dos Estados 5001, 09210-580 St André-SP, Brazil*

E-mail: `aefabris@gmail.com`

Abstract. We present many algorithmic improvements in our early region filling technique, which in a previous publication was already proved to be correct for all connected digital pictures. Ours is an integer-only method that also finds all interior points of any given digital picture by displaying and storing them in a locating matrix. Our filling/locating program is applicable both in computer graphics and image processing.

1. Introduction

Until recently, two classical algorithms predominated as definite choices to implement either point location within a given digital picture, or the filling-up of its inside. The first is called *Seed Fill* [1, 2] and the second *Scan Conversion* [3, 4]. Their strategies are illustrated in Figures 1 and 2, respectively.

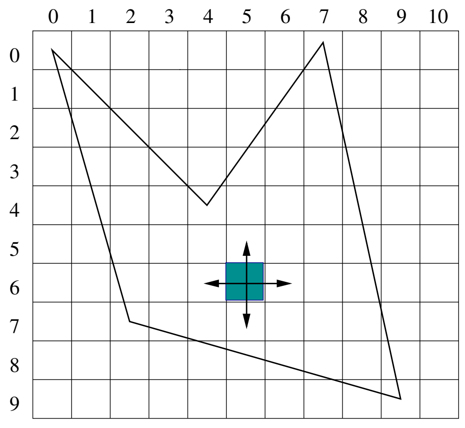

Figure 1: Seed Fill.

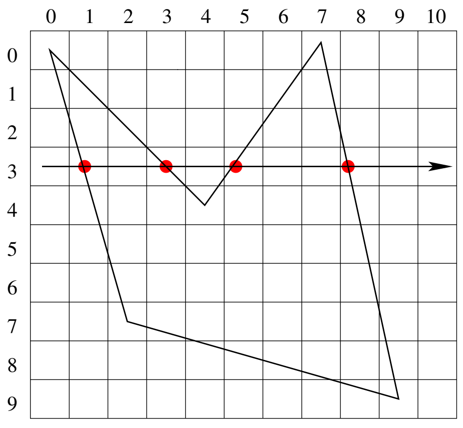

Figure 2: Scan Conversion.

Seed Fill starts from painting a given pixel called *seed* and then all of its subsequent neighbour pixels until meeting boundary conditions. Scan Conversion considers each scan line and determines its overlap intervals within the region by means of pixel parity. The weak and strong points of these algorithms were already discussed in Section 2 of [5] and therefore will be omitted here.

But none of them is free of failures, and these are caused by the following characteristic they have in common: the use of *continuous* Euclidean Geometry constructions within the finite *discrete* space of the computer. Of course, at implementing these classical algorithms one can add several strategies to reduce failures, but challenging regions like nearly degenerate polygons will always prove that they never achieve perfection for all cases. Needless to test complicate regions, in [6] the author presents many *simple* polygons that represent German States, together with a list of failures of the most known implementations of both Seed Fill and Scan Conversion.

Some minor failures can be often dismissed in many circumstances, but not always. For instance, microchips are produced in millions out of a single mould. Just one flaw in the mould will cause a great loss in all senses: time, money, etc. In [5] we illustrate this example and some others in which perfect filling is crucial. There we introduced an integer-only algorithm called CoTRA&FUA that finds all interior points of any given digital picture *exactly*. They are displayed and stored in a locating matrix, hence CoTRA&FUA is *correct* as both filling and locating procedure.

It works for absolutely *any* connected digital picture, no matter how complex or intricate it might be. It is applicable in Computer Graphics and Image Processing even without penalizing speed, for its complexity is quasi-linear.

In this present paper we show many improvements in the CoTRA&FUA algorithm. Among other achievements, we rewrote its implemented source code to have just 193 lines, which represents an approximate 20% reduction compared with the original version of [5]. Moreover, CoTRA&FUA now count on internal procedures that gave much more clarity to the original program. Details are given in the next section, but for the readers who already want to test our program it is available in the link *Codes* of

`https://sites.google.com/site/aefabris`

in Matlab p-code. In Section 7 of [5] the reader will find instructions to run it, together with some examples.

2. Methodology

First of all we need to explain that the input of our program is just an image file in either TIF or JPG format. It should contain a rectangular *canvas* of only black and white pixels. It could have been scanned or created by a mathematical model, but what originated it is unimportant.

Then we *construct* a mathematical model that results in the given canvas, perform the region filling and finally generate the filling matrix. This matrix consists of three different entries: points of the interior, exterior and of the picture itself. In our works they are always graphically painted in yellow, white and black, respectively.

Since no mathematical model is given beforehand, then we disambiguate situations like the ones illustrated in Figure 3 by means of a a precise definition of *exterior* of a digital picture.

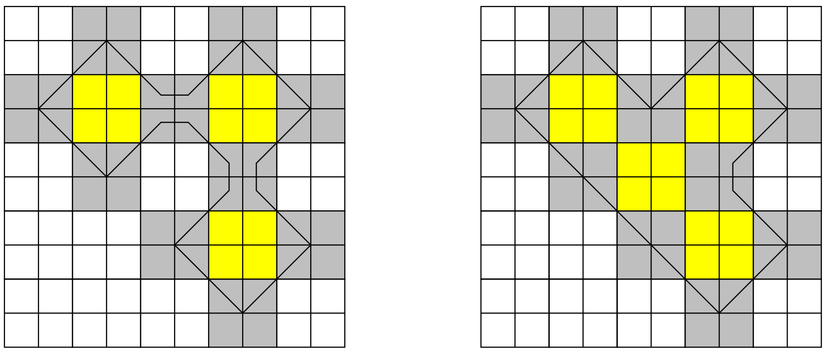

Figure 3: Different curves that generate the same black-and-white digital picture (from [5, Fig. 3])

Technical definitions are given in [5]. Now, CoTRA&FUA are both abbreviations of a double-barreled algorithm. The first stands for *Connectivity and Thickness Reduction Algorithm.* It obtains the so-called *Lego curve* L out of the digital picture. The second stands for *Filling Up Algorithm*, which is always exact for any Lego curve, and its interior coincides with the one of the digital picture. See details in [5].

Now, the main improvements we have got in this present work are the following: **1.** Addition of the internal procedures *Enlarge* and *Tighten* to CoTRA; **2.** Reduction of the code by working only with *thin* instead of *locally thin* curves (Definitions 3.6 and 3.7 of [5], respectively); **3.** Simplification of the part of the code devoted to the *stalk problem* (not explained in [5]);

Regarding **1**, the procedures *Enlarge* and *Tighten* simplified CoTRA because some of its parts had to be repeated in order to replace false *L-pixels* with true ones. Figure 4 illustrates this problem. It shows a Lego curve in blue and magenta, this latter to enhance its L-pixels (Definition 5.4 of [5]).

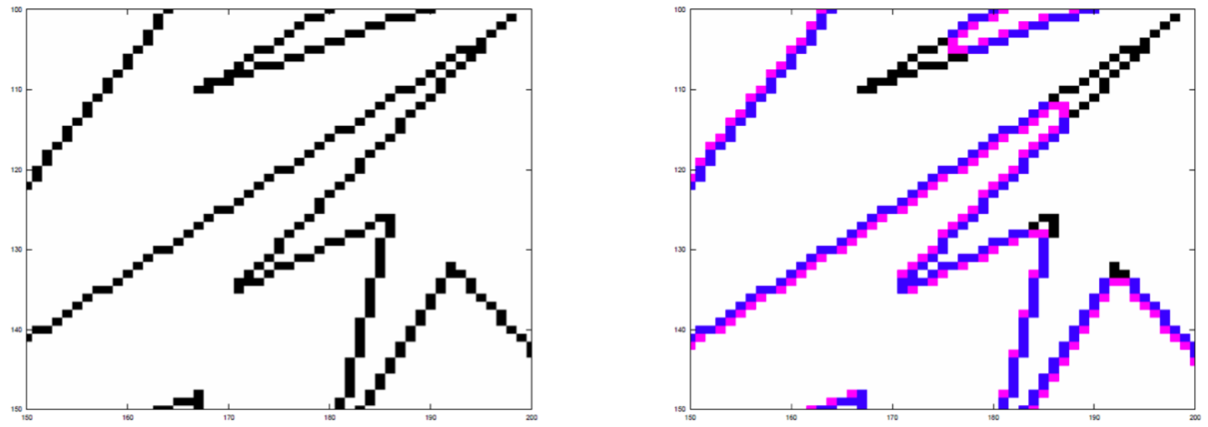

Figure 4: Zoomed input curve (left) and its Lego curve with false L-pixels (right, [5, Fig. 8]).

Roughly saying, *Enlarge* replaces each input pixel p by a 4×3 rectangle of pixels of the same colour of p (black or white), and *Tighten* reverts this process. On the one hand, this introduces an "a priori" multiplication of the computational time by 12. On the other hand, an enlarged input curve always corresponds to a thin Lego curve. Consequently, it is already free of false L-pixels and the problem depicted in Figure 4 does not occur. Then CoTRA runs much much faster, and this explains the improvement **2**.

Regarding **3**, first we need to explain what the *stalk problem* is. For example, consider that our digital picture D has a pixel p with a *unique* neighbour $q \in D$, and q is a *diagonal* neighbour of p. After enlarging D, p will become a black 4×3 rectangle that we call "apple", depicted at the left bottom of Figure 5. The stalk is its connection with a diagonal pixel originated from q, this one also expanded to a 4×3 matrix of black pixels. Now CoTRA marks the Lego curve L as in Figure 6, and there we see a 2×2-square S of diagonal blue and magenta pixels.

Now L is *not* thin because of S, and then FUA will mark internal pixels incorrectly after passing through S. There are some other circumstances in which the *stalk problem* can happen, and all of them were already known in [5]. But there we got round this problem by means of too complicated a strategy, which was then not explained in that work.

Our present version follows a new and much easier strategy: it just makes two copies of the enlarged D with its Lego curve, then shifts right one copy and left the other, and finally overlaps them in order to get an thin Lego curve, as shown in Figure 7.

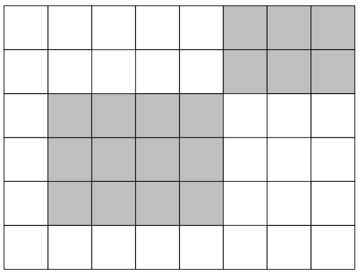
Figure 5: Apple and stalk.

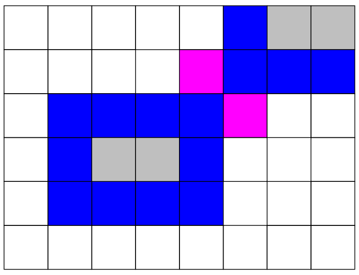
Figure 6: Coloured Lego curve.

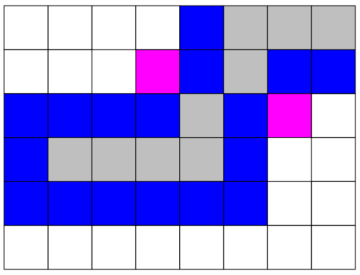
Figure 7: New strategy.

This new strategy also works faster than our previous version in [5]. Together with the fact that we now work only with thin Lego curves, computational time has in fact remained almost the same. Table 1 of [5] summarizes the computational time of 8 tests performed in both Matlab and Octave. It also applies to our new version since there have not been significant changes in time performance.

3. Results

We now have a much simpler and shorter filling/locating program compared with our previous version of [5]. The program is correct for all connected digital pictures, and one of the main reasons for that is our integer-only approach.

4. Conclusions

Our new filling/locating program requires more memory because dimensions are now multiplied by 12. However, modern computers do not have space complexity as a limitation any longer. Hence, increasing space complexity to the detriment of the algorithm is in fact harmless. Time complexity remained the same, but we have gained in clarity and conciseness.

References

[1] Fishkin K P and Barsky B A 1984 *ACM SIGGRAPH Computer Graphics* **18** 235–244
[2] Tsai Y H and Chung K L 2000 *Computer & Graphics* **24** 529–537
[3] Codrea M C and Nevalainen O S 2005 *Computer & Graphics* **29** 441–450
[4] Haines E 1994 ed Heckbert P S (San Diego, CA, USA: Academic Press Professional, Inc.) chap Point in Polygon Strategies, pp 24–46 ISBN 0-12-336155-9 URL `http://dl.acm.org/citation.cfm?id=180895.180899`
[5] Fabris A E and Ramos Batista V 2014 *Aditi International Journal of Computational Mathematics* **3** 1–23
[6] Schirra S 2008 *Algorithms-ESA 2008* (Springer) pp 744–755