Automatic reassembly of irregular fragments

Helena Cristina da Gama Leitão and Jorge Stolfi

Relatório Técnico IC–98-06

Abril de 1998

Automatic reassembly of irregular fragments

Helena Cristina da Gama Leitão *and Jorge Stolfi †

Abstract

This report addresses following problem: given one or more unknown objects that have been broken or torn into a large number of irregular fragments, find the pairs of fragments that were adjacent in the original objects.

Our approach is based on information extracted from fragment outlines that is used to compute the mismatch between pairs of pieces of contour. For that, we use the technique of *dynamic programming*. In order to asymptotically reduce the cost of matching we use *multiple scale* techniques: after filtering and resampling the fragment outlines at several different scales of detail, we look for initial matchings at the coarsest possible scale. We then repeatedly select the most promising pairs, and re-match them at the next finer scale of detail. In the end, we are left with a small set of fragment pairs that are most likely to be adjacent in the original object.

*Departamento de Computação,Universidade Federal Fluminense, 24210-130 Niterói, RJ

†Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

(this blank intentionally left on the page)

Contents

1 Introduction

The subject of this work is the computer-assisted reconstrution of unknown objects that have been broken or torn into a large number of irregular fragments.

We are concerned here with the first step in the reconstruction *fragment-matching*, which is to identify pairs of fragments that were probably adjacent in the original object. When one is dealing with hundreds or thousands of fragments, these pairs may be impossible to find by hand. We develop here a set of mathematical techniques and algorithms that can be used to locate such pairs. We also describe our implementation of those techniques, and the results of preliminary experiments.

After identifying a likely set of adjacent fragment pairs, we will have to face the problem of *global reconstruction*, where the goal is to reconstruct the topology and approximate geometry of the fracture network. We did not address this problem, because its difficulty and practical relevance depend a lot on the correctness and completeness of the list of adjacent pairs which are still being improved upon. Moreover, the global reconstruction step requires human intervention, since it must take into account certain general information that cannot be easily quantified—such as fragment texture and color, decoration, location, probable function, etc. Therefore, we felt that automating the global reconstruction problem would be premature at this point.

1.1 Motivation

Fragment assembly is a very real problem in archaeology, as this quote shows:

> 'Pottery forms one of the most common and abundant types of artifacts found at archaeological sites. As such, pottery provides important information about chronology, technology, trade, and art. Although most handbooks illustrate whole or nearly complete vessels, the majority of ceramic remains from any excavation consists of thousands of small, broken pieces which need to be cleaned, sorted, counted, weighed, and analyzed. [13]'

Besides pottery, the techniques we will describe could be applied to other kinds of archaeological materials, such as glass objects [16], chipped stone tools [13], clay tablets [14], mural paintings [11], ancient manuscripts [3], statues and bas-reliefs [14], building stones, and so on. Our techniques may be useful also in fragment reassembly problems from other disciplines, such as paleontology (fossils), surgery and forensics (shattered bone) and failure analysis (debris).

1.2 Related work

At present, the reconstruction of archaeological fragments is done largely by hand; computers are used, if at all, only in the classification, indexing, and presentation of scanned images of the fragments [3, 11, 29], which are indexed and retrieved based solely on textual descriptions provided by the user. There is a modest technical literature about image enhancing techniques specialized for archaeological materials [29, 11]. Only a few authors have

considered the use of computer vision and pattern matching techniques to automatically extract indexing information from digital images of archaeological artifacts [28].

The specific problem of identifying adjacent ceramic fragments by matching the shapes of their contours was recently considered by Üçoluk and Toroslu [28]. Their algorithm considers only a fixed scale of resolution, and therefore has large expected asymptotic cost (see section 2.6). No experiments are reported in the paper.

The fragment reassembly problem is similar to that of *automatic assembly of jigsaw puzzles*, which has been addressed before mainly as a cute exercise in robotics and machine vision [8, 15, 7, 25]. In particular, H. Wolfson and G. C. Burdea developed a program that can find matching pieces in a standard puzzle game [7], and even control a robot arm to assemble the puzzle. However, the techniques used rely on special characteristics of puzzle games (smooth borders and well-defined corners), which are not found in archaeological materials.

More generally, the problem can be viewed as a special case of *approximate contour matching*, an area with extensive bibliography [18, 22, 15, 17, 5, 24, 32, 1, 20, 26]. In particular, there are many papers on machine vision that are concerned with *object recognition from outlines*, mainly for industrial or military applications. Some of their results can be expected to work also for irregular fragments. However, most of these works assume that the outlines are to be matched agaist a relatively small set of fixed templates, so that their main concern is to reduce the cost of comparing an outline against a template. In the application that interest us, however, the 'templates' are the fragments outlines themselves, which may number in the thousands. We are thus interested mainly in techniques that can be used to eliminate large sets of templates at relatively low cost.

2 Approach

2.1 The ideal model for fracture objects

Our approach is based on information extracted from fragment outlines. The idealized model can be summarized as follows. We assume that the fragmented objects have a well defined smooth surface. This surface is divided into two ou more parts, the *ideal fragments*, that are separated by *ideal fracture lines*, irregular curves with zero width. Two fragments are said to be *adjacent* if they share a fracture line. The fractures also split the original outline of the surface into one or more *border lines*.

The fracture lines can be viewed as a graph G drawn over the object's surface, which we call the *fracture network*. See figure 1(a). The point where three or more lines (fracture or boundary) meet is called an *ideal corner*. The boundary of an ideal fragment is an *ideal contour*; it is the concatenation of one or more fracture and border lines.

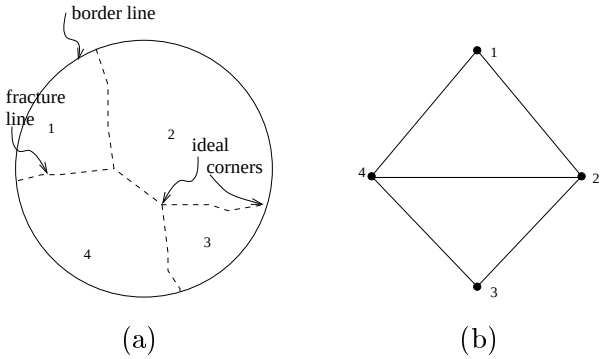


Figure 1: Ideal fracture network (a) and the adjacency graph (b).

We can assume that each fragment is a topological disk, that is, its contour is a single loop. In the cases where this assumption does not hold (for example, when a handle or cylindrical vessel is broken at both ends), we can pretend that each closed loop in the boundary of the fragment is the contour of a separate fragment. The same holds for objects with two smooth surfaces of similar characteristics (torn paper, ceramic pottery and tiles, etc.): we can view the two sides of each fragment as two separate fragments. The physical connection between these two contours only needs to be taken into account at the very last stage of the reconstruction (which does not concern us here).

The topological dual of the fracture network (ignoring the border lines) is the *fragment adjacency graph* G^*. It has one vertex u^* in each fragment u; for each fracture line e, separating two fragments u and v, there is an edge e^* in G^* connecting u^* and v^*, which crosses the boundary of u only once, at some point of e. Note that while G is both a geometrical and topological object, G^* is topological object only.

2.2 The actual data

The fracture lines and ideal contours are mere abstractions, of course. What we obtain from the physical fragments are the *observed contours*; they differ from the ideal contours for a number of reasons, such as barbs (common in torn paper), turned edges, or the loss of small portions of material. Because of these perturbations, the single fracture line that separates two adjacent ideal fragments becomes two slightly different sections of the corresponding observed contours. This difference is where lies the main difficulty of the reassembly problem.

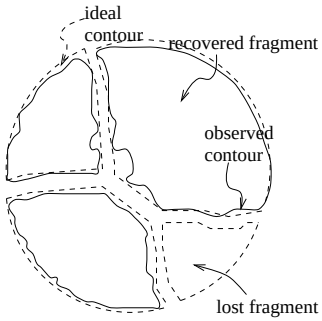


Figure 2: Observed contours.

A *segment* is a piece of contour. A pair of *corresponding segments* is a pair of maximal segments, belonging to two observed contours, that correspond the same ideal fracture line.

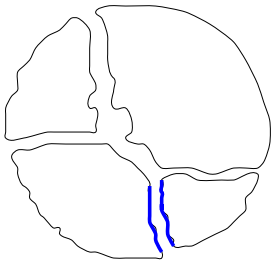
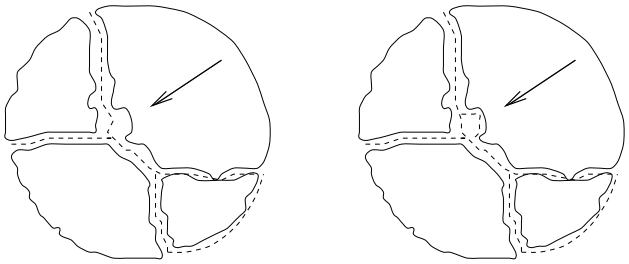


Figure 3: Corresponding segments.

2.3 Geometric features of obtained contours

In typical applications, we can expect that there are *lost fragments*. Therefore, there is an intrinsic ambiguity in this model: the loss of a small portion of the object along a fragment boundary can be either attributed to noise (figure 4(a)), or modeled as an ideal fragment that was not recovered (figure 4(b)). Our algorithms will arbitrarily resolve this ambiguity one way or the other, depending on the size of the missing piece.



(a) (b)

Figure 4: Ambiguity of the fracture model.

Another ambiguity of the model is that it is not possible to define with precision the beginning and end of corresponding segments. For one thing, we do not know the position of

the ideal fracture corners; and, in any case, the presence of noise does not allow us to define a precise point-to-point correspondence between the observed and ideal contours. Again, our algorithms will arbitrarily define the beginning and end of corresponding segments, based on the distance between the two contours.

An important property of physical fracture lines is that each ideal corner is generally incident to only three fracture lines, or one fracture line and two border lines. Moreover, two of these lines frequently make an angle close to 180°; that is, they are actually parts of the same physical fracture. Thus, the existence of the corner will be apparent only in two of its three incident fragments. Moreover, physical fracture lines tend to display sharp bends even where there are no ideal corners. For these reasons, we cannot expect to identify the contour segments to be matched by looking for corners in the fragment outlines.

2.4 Reconstruction goal

We formulate, the problem of *fragment matching* as follows: given a set $\mathcal{F}$ of observed contours, determine as much information as possible about the ideal fracture graph G. In particular, try to determine all pairs of adjacent fragments, and the corresponding segments of their contours.

2.5 Identity of contours

The first question we must answer is whether the problem can be solved at all. Is it really possible to identify adjacent pairs in a large collection of fragments, only on the basis of the (noisy) observed contours? The answer is yes: by physical experiment, one can verify that the irregular nature of fracture lines allows pairs of adjacent fragments to be identified, with high confidence, by the congruence of their outlines at sub-millimeter scale. See figure 5.

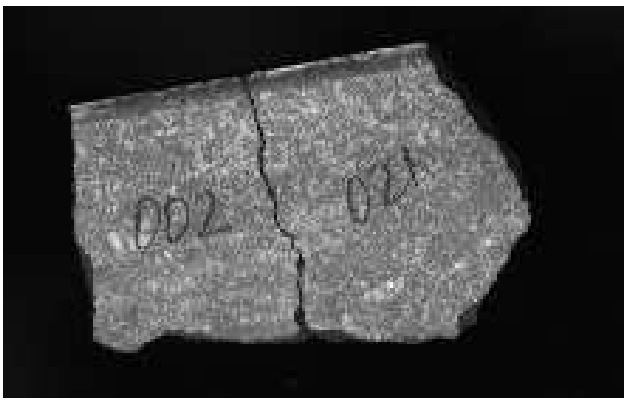


Figure 5: Correspondence of fragments

Our strategy to solve the reconstrution problem exploits this observation. If two contour segments are sufficiently long, and their shapes are sufficiently similar, we can discard the hipotesis that the similarity is due to mere coincidence. In section 3.2, we will analyze the relation between measurement accuracy, noise magnitude, and the probability of correctly identifying two corresponding segments of given length.

2.6 Computational dificulties

From the computational viewpoint, the main dificulty of this problem is the large number of segment pairs that we would need to consider.

In order to identify adjacent fragments with adequate confidence, we need to work with contour segments comprising hundreds of independent sample points. Moreover, for each pair of fragments, we would need to test all possible pairs of segments. As we will see in section 9.1, if we have N fragments, each fragment with L sample points, then exhaustive comparison will require $O(N^2L^4)$ operations. In typical instances of the problem, with thousands of fragments, this exhaustive search would be way too expensive.

In order to asymptotically reduce this cost, we will use two complementary ideas. First, we decrease the number of sample points in each contour, by filtering and resampling. Second, we reduce the number of segment pairs to be compared in detail, by using *multiscale* techniques [2, 21]. In section 9, we show how these techniques allow us to reduce the cost to $O(N^2 + ML^2)$, where M is the number of adjacent fragment pairs.

By using *geometric hashing*, as proposed by Kalvin and others [15], it would be possible to reduce the term $O(N^2)$ to something closer to $O(N \log N)$. We did not implement this optimization yet.

2.7 Stages of processing

Our solution consists of the following stages:

- Acquisition of fragment images;
- Image segmentation and contour extraction;
- Filtering of the contours at different resolution scales;
- Encoding of the contours as curvature strings;
- Statistical analysis of the contours;
- Identification of similar segments at the coarsest scale;
- Refining of candidate matchings at finer scales;
- Geometric alignment of similar segments;
- Graphical presentation of results.

In the following sections, we describe each stage in more detail.

3 Fragment image acquisition

The acquisition of fragment images can be done of many ways, depending of nature of the objects. In the case of pieces of paper, for example, we can directly use a flatbed scanner, as was done in figure 6.

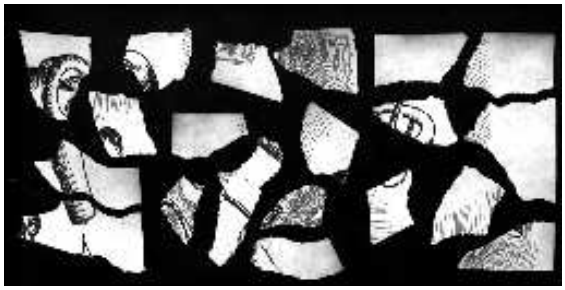


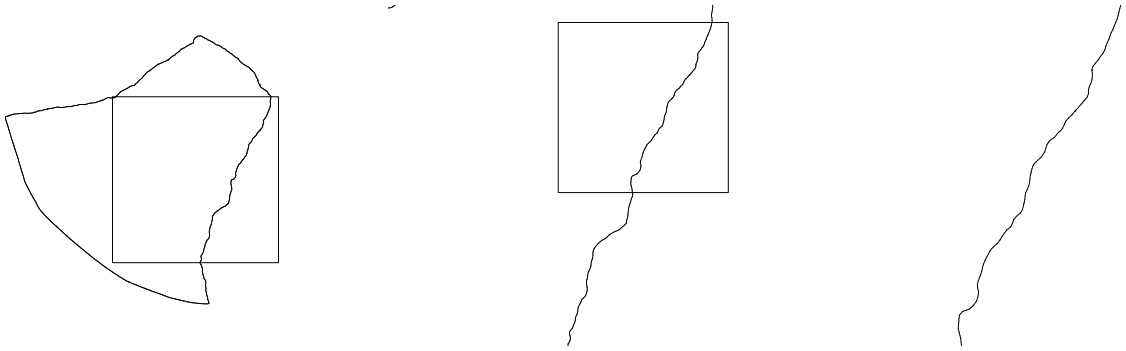
Figure 6: Pieces of paper.

For other objects with a flat surface, such as tiles and mural paintings,the direct use of a scanner may still be possible, although a photographic or TV camera may be more adequate for bulky or delicate materials.

In the case of pottery and stone fragments, where the contours to be matched are three-dimensional curves, it would be necessary to acquire two or more images of each fragment, from differents angles, and use techniques of stereoscopic vision to extract the contours. In this report, we consider only the case of flat surfaces.

3.1 The geometry of fracture lines

For the aplications we are interested in, fracture lines are self-similar to a certain extent; that is, a small portion of a contour, magnified, is statistically similar to the whole. See figure 7. Indeed, fracture lines are a classical example of *fractal curves* [12]. It is precisely their randomness at many different scales that makes our multi-scale approach possible.



(a) (b) (c)

Figure 7: Fractality of fracture lines.

The fractal model is apropriate for fractures in rigid and granulated materials, such as ceramic, stone, and stucco. The fractal model can also be applied to certain types of aged paper (which tend to crumble rather than tear).

Fractures in glass or glazed ceramic objects tend to be smoother (technically, their fractal dimension is closer to 1); on the other hand, the observed contours are much sharper, and therefore compared with greater precision.

3.2 Minimum resolution

The resolution of scanned fragment images must be sufficiently fine to capture the small irregularities which are the key to the identification of adjacent fragments.

On a first aproximation, if we have N fragments, and we want to identify with high confidence the partner of a determined fragment, then we need to extract at least $\log_2 N$ bits of information from the shared part of their contours. In the case of irregular fractures, we can assume that the shape of the scanned contour gives on the order of one bit of information per linear pixel. Therefore, the resolution of the scanner should be at least $\log_2 N/L$ pixels per unit of length, where L is the length of the shared segment.

For example, if we have $N = 1000$ fragments, and we want to identify pairs that share at least one inch of their outline, we must obtain images with at least $\lceil \log_2 N \rceil = 10$ pixels/inch resolution. In pratice, since image scanning is not the only source of error, we will need to scan the fragments at several times this resolution.

It is important to observe that the resolution required increases only logarithmically with the number of fragments. In other words, it is theoretically possible to identify the adjacent fragments, with high confidence, even in large instances of problem (with millions of fragments), from contours acquired at ordinary scanner resolutions.

4 Identification and separation of fragments

For obvious practical reasons, one often scans two or more fragments at a time. Therefore, the first image processing problem we must solve is that of identifying and separating the individual fragments in each image.

In the algorithms of this section, we will use the following notation. The variables n_x and n_y denote the horizontal and vertical dimensions of the scanned image, in pixels. $R = [0..n_x - 1] \times [0..n_y - 1]$ is the domain of the image. A generic pixel position p is therefore a pair $(p_x, p_y) \in R$. The value (intensity) of the image at pixel p is $v[p]$.

To identify the fragments, we use a simple thresholding algorithm, that assumes that there is a unique grey level δ_{sep} such that pixels with color $v[p] \leq \delta_{\text{sep}}$ belong to the background and pixels with color $v[p] > \delta_{\text{sep}}$ belong to the fragment. This simple algorithm should be adequate in the case of thin flat fragments, since we can usually choose the background color at the time the images are scanned. We will assume, therefore, that the background has a uniform color $\delta_{\text{min}} < \delta_{\text{sep}}$.

Once we have located a pixel $v[p] > \delta_{\text{sep}}$, we use a straightforward propagation algorithm (a variant of the flood-fill technique that is used in many graphic editors to paint region of arbitrary contours) to identify all pixels q with $v[q] > \delta_{\text{sep}}$ that are conected to pixel p, in the 4-neighborhood topology [23].

We also extract a one-pixel wide safety "halo" around the fragment. These extra pixels are essential for the precise positioning of the fragment contour, as will be explained in section 5. We assume here that the fragments are sufficiently separated from each other and from the image border, so that these safety halos do not overlap, and surround the entire fragment.

The output of this stage is a set of images, each containing a single fragment and its safety halo, against a uniform background of color $\delta_{\min}$. See figure 8.

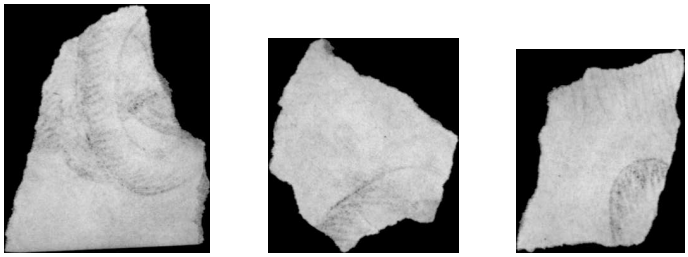


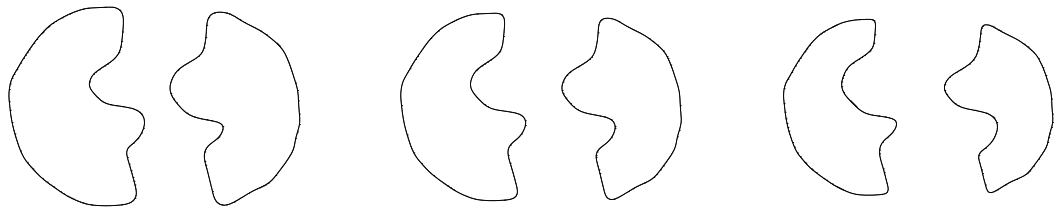
Figure 8: Examples of separated fragments.

5 Extration of contours

Once we have a separate image for each fragment, we extract its raw contour, in the form of a polygonal closed curve. As explained above, we assume that the fragments were scanned against a uniform background with known color $\delta_{\min}$. If each fragment has an approximately uniform color $\delta_{\max}$, like the fragments shown in 8, the contour can be extracted by a simple algorithm that follows the level curve of an intermediate intensity δ_{med}, somewhere between $\delta_{\min}$ and $\delta_{\max}$.

In fact, it is important that the threshold δ_{med} be exactly $(\delta_{\min} + \delta_{\max})/2$, in order to ensure that the contours of adjacent fragments are congruent. This requirement can be deduced from the assumptions that the scanned images are linear blurrings of the ideal (infinite-resolution) images, and that the original (unbroken) object had uniform color $\delta_{\max}$ on both sides of the fracture.

Figure 9 shows the effect of varying the threshold δ_{med} on an image of two small congruent fragments. The background intensity $\delta_{\min}$ is 10/255, and the fragment color $\delta_{\max}$ is 200/255.



$\delta_{\text{med}} = 30/255$ $\delta_{\text{med}} = 105/255$ $\delta_{\text{med}} = 170/255$

Figure 9: Thresholding efects.

Observe that the extracted contours are congruent only when δ_{med} is near the average of $\delta_{\min}$ and $\delta_{\max}$. An incorrect choice of the threshold δ_{med} will change the curvature of convex and concave parts in oposite directions, hampering the recognition of corresponding contours.

5.1 Contour extraction algorithm

To extract the contour of a fragment, we first locate a pair of adjacent grid points, (a, b), with a inside the fragment ($v[a] > \delta_{\text{med}}$), and b outside ($v[b] \leq \delta_{\text{med}}$). We then follow the fragment boundary in counterclockwise sense while maintaining these conditions.

More exactly, at each step the algorithm can be in one of the situations shown in figure 10:

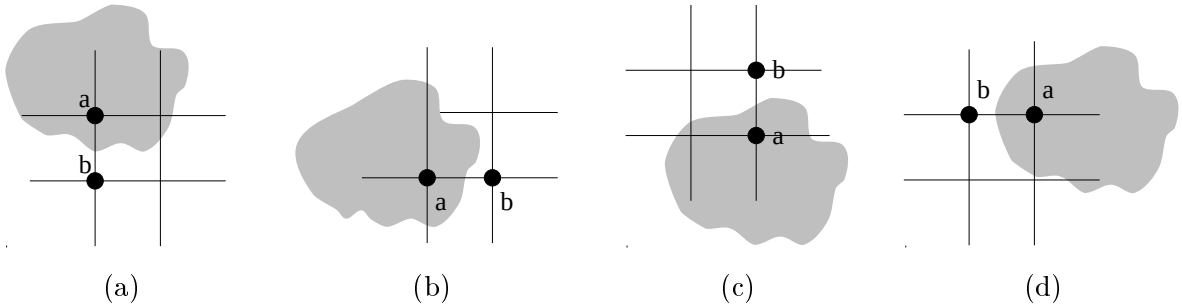


Figure 10: Possible situations during contour extraction.

Suppose that a is directly above b, as at figure 10(a). From this situation, the algorithm can advance either a or b, or both, so as to keep a inside and b outside the fragment, as shown in figures 11(a–c).

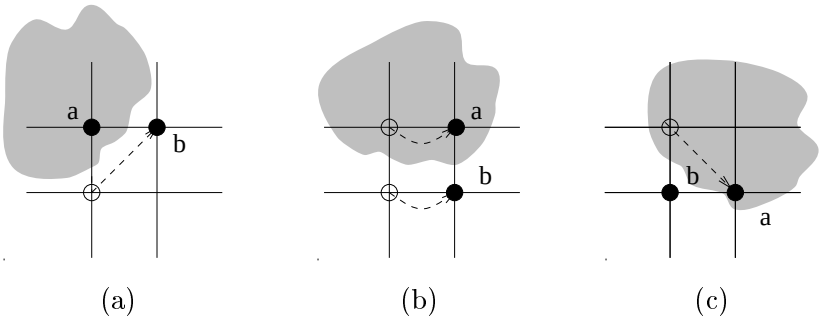


Figure 11: Alternative steps of algorithm for the situation of figure 10(a).

The handling of the other cases (figures 10(b–d)) is analogous.

We obtain from this process a sequence of pairs (a_i, b_i), where a_i follows the contour from the inside (i.e. $v[a_i] > \delta_{\text{med}}$), and b_i from the outside (i.e. $v[b_i] \leq \delta_{\text{med}}$). For greater precision, from each pair a_i, b_i we compute the (fractional) point p_i on the edge (a_i, b_i) where the linearly interpolated grey level is exactly δ_{med}. (See figure 12)

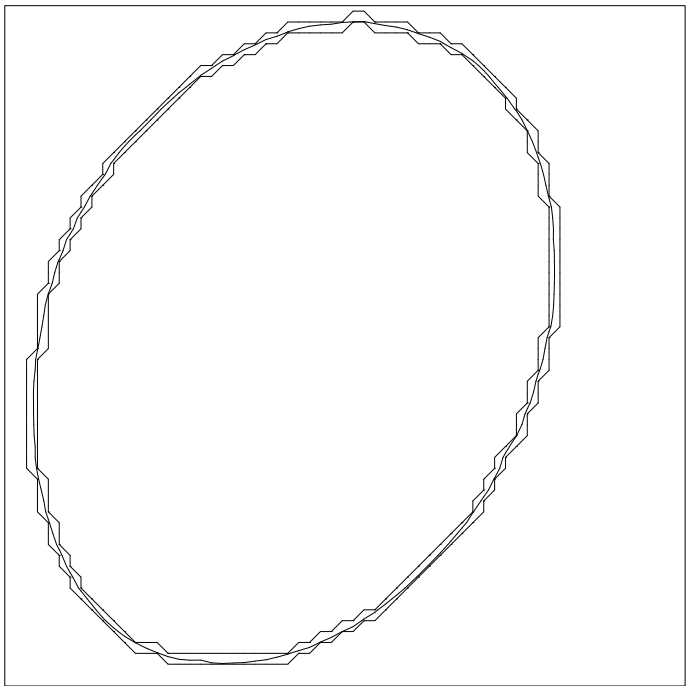


Figure 12: Extracted contours: external, interpolated and internal.

The result of contour extraction is, therefore, a closed polygonal curve $p_0, p_1, .. p_n = p_0$. The sides of this polygonal have variable length between 0 and $\sigma\sqrt{2}$, where σ is the distance between consecutive pixels.

This simple contour extraction algorithm has its limitations. Even for flat and thin fragments, such as the torn paper of figure 6, the algorithm is frequently confused by barbs, turned edges, loose threads, etc.—like those that are visible on the right side of the fragment in figure 8(a).

Another serious shortcoming of this algorithm is the fact that, in pratice, the foreground color $\delta_{\max}$ changes a lot from point to point; therefore, there is not a unique correct value for the thresholding δ_{med}. For such applications, one should use a more sophisticated segmentation algorithm that varies $\delta_{\max}$ (and hence δ_{med}) from pixel to pixel, based on the image values on the neighboring pixels.

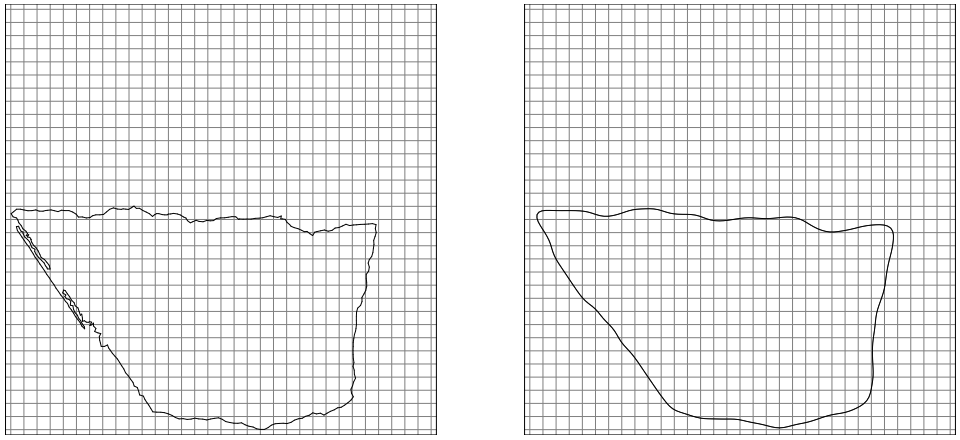
5.2 Re-sampling

The irregular spacing of the points p_i in the intermediate contours is problematic for the filtering stage. Therefore, before storing the extracted contour, we re-sample it with roughly uniform steps, equal to approximately half the pixel spacing, using cubic interpolation [4].

6 Filtering

The filtering phase takes as input the observed contours, and outputs smoothed versions of them. See figure 13. The filtering serves two different purposes. Firstly, it is used to remove

non-significant irregularities from the contours, caused by artifacts of the image acquisition and contour extration processes, or by erosion of the material after fragmentation. For this purpose, it is necessary to remove the highest-frequency components of the contour, because they are the ones more affected by these perturbations. Secondly, the filtering is used to reduce the complexity of the contours, for the purpose of speeding up the identification of similar contour segments. This use will be further explained in section 9.

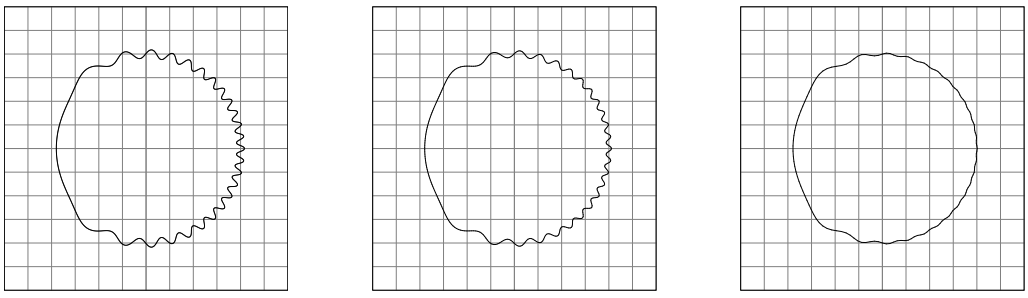


(a) Original curve. (b) Filtered to scale $\lambda = 64$ pixels.

Figure 13: Filtering a curve. (Grid spacing = 25 pixels apart.)

Filtering time-based signals is a well developed field with extensive theory and effective, well understood procedures. Filtering a curve is harder since the natural parameter for describing the curve, its length, may shrink substantially and non-uniformily as the result of filtering. To solve this problem, we developed an original technique which we call *geometric filtering*, to be detailed in a separate report [10].

The goal of filtering is to remove all details smaller than a certain size λ, the *filtering scale*. See figure 14.



(a) $\lambda = 8$ pixels (b) $\lambda = 16$ pixels (c) $\lambda = 32$ pixels

Figure 14: Filtering a curve at different scales (and spacing = 25 pixels)

6.1 Final sampling

After filtering, each contour is resampled at uniform steps along the curve. The number n_s of samples that we use is chosen based on the Nyquist criterion [6]. Namely, for a contour of length L that has been subjected to geometric Gaussian filtering at scale λ, we use

$$n_s = \left\lceil \frac{8L}{\lambda} \right\rceil$$

This formula is justified by noting that Gaussian filtering at scale λ effectively removes all Fourier components of the curve with wavelength smaller than $\lambda/4$ [10].

The corresponding step size is

$$d = \frac{L}{n_s} = \frac{L}{\lceil \frac{8L}{\lambda} \rceil} \approx \frac{\lambda}{8}$$

Since n_s is usually in the hundreds range, the sampling step d is actually quite close to $\lambda/8$, and therefore practically the same for all contours filtered at the same scale λ. Our contour matching algorithms will take advantage of this fact.

7 Generic curve comparison

7.1 Basic concepts

7.1.1 Segments and candidates

In this section we will consider the problem of comparing the shape of two curves in a generic context. We will assume that each contour is represented by a circular sequence of *samples* $c_0, c_1, .. c_{n-1}$. Each sample can be a point of the contour, or any other local geometric property such as curvature or color. We assume also that the samples are uniformly spaced along the contour, and that the sampling step d is approximately the same for all contours.

We will need the following concepts:

A *step* is a pair of consecutive samples c_i, c_{i+1} on the same contour.

A *segment* is a sequence of one or more consecutive samples of a contour, strictly less than the complete contour. We denote by $\langle c, i, m \rangle$ the segment of contour c that consist of the samples $c_i, .. c_{i+m-1}$ (all sample indices are implicitly taken modulo the number of samples in the whole contour).

A *candidate* is a pair of segments from two different contours.

7.1.2 True and recognizable candidates

A candidate is said to be *true* if its segments correspond to the same ideal fracture line, and differ only by image acquisition errors or loss of very small intervening pieces. Otherwise the candidate is *false*.

The concept of true candidate is not very useful for the purposes evaluating algorithms, because it is based on information that may not be present in the input data. If the material loss and acquisition errors are large enough, the two segments of a true candidate may have completely different shapes. Moreover, the true candidates include also pairs of segments whose ideal fracture lines are too small to allow unambiguous matching.

Therefore, we are only interested in a true candidate if it is also *recognizable*; meaning that it can be identified by a careful and patient person, *working only with the extracted contours*.

7.1.3 Solutions and quality measures

In general, a *solution* for the fragment reconstrution problem is a set of candidates. Ideally, the solution should be precisely the set R of recognizable candidates; but we cannot yet expect the computer to perform as well as a human. In order to evaluate the quality of a solution S, we define the following measures:

$$\textit{Sensitivity}: \quad \nu = \frac{|S \cap R|}{|R|}.$$

$$\textit{Significance}: \quad \gamma = \frac{|S \cap R|}{|S|}.$$

Informally, The sensitivity ν measures the ability of the program to recognize true candidates, while the significance γ measures its capacity to reject false ones.

7.2 Evaluating a candidate

We will try now to define a quality measure for a single candidate, that is, a measure of how different the two segments are — in shape, or in some other derived characteristic, such as curvature or color.

Basically, the mismatch is can be defined as an integral of the distance between corresponding points along the two segments. The main difficulty here is to establish a likely correspondence between each point of one segment an the point on the other.

7.2.1 Segment reversal

When we extract the contour of a fragment, we always follow its boundary counterclockwise, as explained in section 5. Therefore, two segments that come from the same fracture line will be traversed in opposite directions, as shown in figure 15.

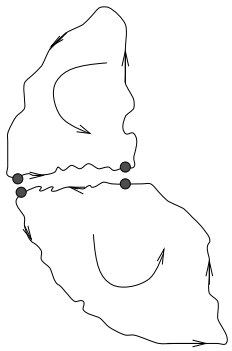


Figure 15: Matching segments are extracted in opposite directions.

In the remaider of this section, we assume that one of the fragments to be compared has been reversed, to account for this fact.

7.2.2 Sample matching

We will define a *discrete matching* of the two segments $a_0 \ldots a_R$ and $b_0 \ldots b_S$ as a sequence of pairs (r_j, s_j), $j \in \{0, .. m-1\}$ satisfying the following conditions:

- $r_j \in \{0, .. R-1\}$,
- $s_j \in \{0, .. S-1\}$,
- $r_j \leq r_{j+1} \leq r_j + 1$,
- $s_j \leq s_{j+1} \leq s_j + 1$,
- $(r_j, s_j) \neq (r_{j+1}, s_{j+1})$

Once established the matching r, s we compute the mismatch between the two segments by the formula:

$$C(a_{0..R}, b_{0..S}) = d \sum_{i=0}^{n-1} (\|a_{r_i}, b_{s_i}\|^2 - \xi^2) \tag{1}$$

where $\|a_{r_i}, b_{s_i}\|$ is a measure of the difference between samples a_{r_i} and b_{s_i}, and ξ is a parameter to be determined, the *comparison threshold*.

The matching we use is the one that minimizes formula (1), among all discrete matchings of the two segments. We find this matching using a standard dynamic programming algorithm [19] modified to allow flexible end conditions. More details are given in a separate technical report [9].

Note that the mismatch $C(\ldots)$ is negative if the sample difference $\|a_{r_i}, b_{s_i}\|^2$ is less than the threshold ξ^2, in the squared-mean sense. If the candidate is sufficiently long, and ξ is the maximum average displacement of observed samples away from the ideal fracture line, then negative and positive values of C indicate true and false candidates, respectively, with very high confidence [9].

8 Geometric invariants

Since each fragment was digitized with arbitrary orientation and position, it is not feasible to find similar pairs by comparing directly the coordinates of sample points. In order to avoid the high cost of testing all possible rotations and translations when comparing two segments, we must extract from them some kind of *geometric invariant* — some shape information that will not change when the fragment is rotated or translated.

The area, length, and moments of inertia are familiar examples of such geometric invariants. Unfortunately, these invariants are not useful for our problem, because they depend on the whole curve. The invariants that we need must be local, this is, they must depend only on a small portion of the contour.

The simplest local invariant is the *curvature*. This invariant has been widely employed in shape recognition work, as for example in the work of Wolfson[31], Rosin[24], Boussofiane et al.[5], and Kalvin et al.[15].

The curvature of a curve c at time t is given by the following expression:

$$\kappa = \frac{|c^{\cdot}(t) \times c^{\cdot\cdot}(t)|}{|c^{\cdot}(t)|^3} \tag{2}$$

where $c^{\cdot}(t)$ is the first derivative, of the curve (the *velocity vector*), and $c^{\cdot\cdot}(t)$ is the second derivative (the *acceleration vector*).

A disadvantage of the curvature is that it is very sensitive to noise [31]. Small changes in the shape of the contour may cause large changes in the curvature. For the same reason, the computation of $c^{\cdot}(t)$ and $c^{\cdot\cdot}(t)$ by numerical differentiation is very sensitive to sampling error. We can reduce the severity these problems by working with the filtered curve.

Another possible problem of the curvature formula (2) is that the denominator can be zero, or too small. This problem does not happen in our case, because our filtered contours are parametrized by arc length, and therefore $|c^{\cdot}(t)| = 1$ for all t.

Apart from computational issues, a disadvantage of using the curvature instead of position is that curves with similar local curvature may have widely different shapes. More precisely, the mismatch of two segments, computed from their curvatures, does not provide any guaranteed bound on the mismatch computed from their shapes. Nevertheless, in pratice we observe that the two measures are strongly correlated.

8.1 Curvature sampling

To avoid aliasing artifacts in the sequence of curvature samples, the sampling frequency needs to obey the Nyquist principle as explained in section 5.2. This bound assumes that the curve has been filtered with Gaussian filter. The latter decays so fast at high frequencies that the $\lambda/4$ cutoff, which we assumed in section 5.2, is still valid—in spite of the extra factor $(2\pi k)^2$ that appears in the Fourier transform of the second derivative.

In fact, since our contour comparison algorithms consider only discrete matchings (where each sample of a curve is precisely aligned with a single sample of the other curve), we must use a higher sampling frequency, high enough so that the error caused by this limitation be tolerable. More precisely, if the rate of change of the curvature $|d\kappa/dt|$ is limited by

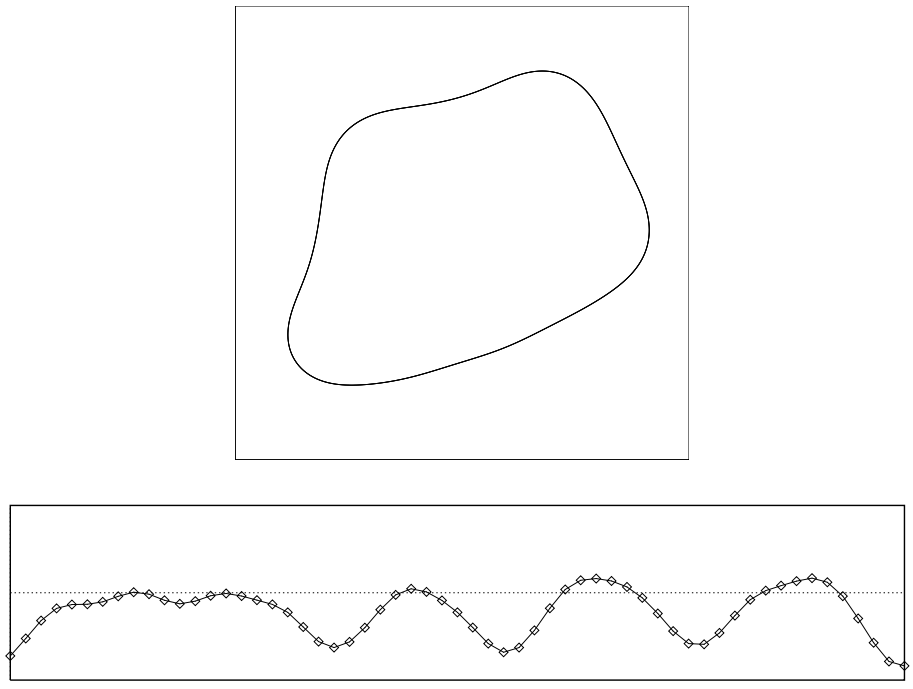
a constant β, and the time interval between two samples is ϵ, then limiting the mismatch computation to discrete matchings is equivalent to perturbing the curvature by an amount no greater than $\epsilon\beta/2$. Therefore, if ξ is a significant difference in curvature, we must use a sampling step $d \leq 2\xi/\beta$.

8.2 Curvature encoding

The curvature samples computed by formula 2 are real numbers. As we will see, the contour comparison step accesses the curvature graphs of all contours in random order, which therefore have to be simultaneously stored in memory. In order to reduce storage requirements, both in disk and in memory, we encode each curvature sample as a single byte.

In the current implementation, we use an alphanumeric encoding, with upper case letters for positive curvature values (convex parts of the contour), lower case for negative values (concave parts), and '0' for zero curvature (straigth parts). (We chose this restricted-range encoding to simplify debugging. In the final implementation, we plan to switch to the full signed byte range $[-127.. + 127]$).

The result of encoding each contour is a *curvature string*, a sequence of one or more symbols from the alphabet `z..a0A..Z`, one symbol per sample 16.



`TROLKKJEaBIKIEAEIKMPRSRQMCfaIMPRSSQLeklkgGNQRSQNHchkljEORTT`

Figure 16: A filtered contour (top), its curvature graph (middle) and its encoded curvature string.

8.3 Quantization function

The curvature values are unbounded in principle, so the mapping to the finite symbol set `z..a0A..Z` must be non-linear. In fact, to maximize the information contents of the encoded values, the curvature values should be normalized so that that they have approximately uniform distribuition over the range $\{-26..+26\}$.

For this purpose, we use the following function

$$\phi(\kappa) = \frac{1}{\delta} \log \left(\frac{\delta\kappa}{\epsilon} + \sqrt{\left(\frac{\delta\kappa}{\epsilon}\right)^2 + 1} \right) \tag{3}$$

where ϵ and δ are two adjustable parameters. The encoded curvature string then consists of the values of $\phi(\kappa)$ at each sampling point of the curve, rounded to the nearest integer, clipped to the range $-26..+26$, and encoded with the symbols `z...a0A...Z`.

Formula (3) is similar to the *μ-law compression* [27] that is often used in signal processing. The function ϕ is approximately linear, with slope $1/\epsilon$, for values of κ near zero; and approximately logarithmic in base δ for large values of $|\kappa|$. See figure (17).

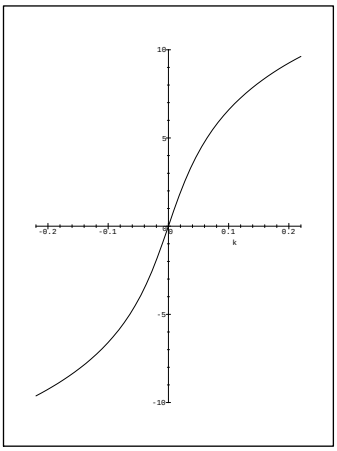


Figure 17: Graph of $\phi(\kappa)$ for $\epsilon = 0.01$ and $\delta = 0.25$.

The reason for choosing this formula in particular is that it allows independent control of the absolute quantization step for small curvature values (which is approximately ϵ) and the relative step for large values (which is approximately δ).

The quantization steps must be small enough to ensure that significant differences of curvature are preserved in the encoded string. That is, the quantization error must be small in comparison with other kinds of noise that affect the curvature graphs. Since most curvature values will fall in the range where ϕ is approximately linear, the quantization error can be modeled as a perturbation uniformly distributed between $-\epsilon/2$ and $+\epsilon/2$, with expected value zero and standard deviation approximately $\epsilon/\sqrt{12}$. Therefore, reasonable choice for ϵ is some fraction of the threshold ξ introduced in section 7.2.2.

Once the value of ϵ has been fixed, we fix the value of δ so that the histogram of $\phi(\kappa)$ becomes as uniform as possible.

Recall that, in the comparison of the segments, one of them must be flipped back-to-front, and the sign of its curvature values must complemented. Since the ϕ function is symmetric around zero, complementing the sign of curvature sample corresponds in our encoding to inverting the case of the corresponding letter.

9 Multiples scales

9.1 Comparison cost

Suppose we have a total of N observed contours with mean length C, and that the distance between two samples is d. The average number of samples per contour is then $n = C/d$ Suppose also that we wish to detect all pairs of adjacent fragments where the shared fracture line has length $C_{\min}$ or greater. In practice, we can assume that $C_{\min} = \beta C$ for some constant β nor much smaller than 1. The minimum number of sampling steps in a candidate is then $n_{\min} = C_{\min}/d = \beta n$.

The cost of comparing two segments with $n_{\min}$ samples each, with the dynamic programming (DP) algorithm, is proportional $(n_{\min})^2$. The number of candidates to compare, for two given contours of n samples each, is about $n^2/n_{\min} = n/\beta$. (There are n^2 pairs of segments of length $n_{\min}$ in the two contours; however, pairs that have similar alignments and a sizeable overlap are in a sense redundant, and it would be possible to skip most of those redundant pairs, without compromising the algorithm's sensitivity.) Since we have N contours, that must be compared by pairs, the total cost would be $\Omega(N^2 n^2 (n/\beta)) = \Omega(N^2 n^3)$.

Recall that we would like to solve problems with thousands of fragments, each one with thousands of samples. Obviously, the cost of this simple-minded approach would be prohibitive.

9.2 The multiscale approach

To reduce the cost of matching, we use a *multiscale technique* [30]. We begin by solving the problem for coarse versions of the contours, sampled with the largest possible step d, namely $d_0 \approx C_{\min}$ and $n_{\min} \approx 1$. At this scale, each contour will have only $1/\beta = O(1)$ samples, so the exhaustive comparison can be done in time $O(N^2)$. Of course, the limited information available will result in a huge number of possible candidates, on the order of N^2.

That done, we solve again the problem at a finer scale, with contours sampled with a smaller step $d_1 = d_0/2$; but now we *compare only those candidates that looked promising in the previous step*. We repeat this process at finer and finer scales, with sampling steps d_r decreasing in geometric progression. At the last stage, when we take into account all the information available in the contours, we should be left with a small set of candidates that have good chances of being true.

In order to avoid aliasing noise, which could lead us to incorrectly reject good candidates at the coarsest scales, the contours we use at each stage must have been filtered so as to

satisfy the Nyquist criterion. Namely, the characteristic length $\bar{\lambda}_r$ of the filter must satisfy $\bar{\lambda}_r \leq 8d_r$.

9.3 Analysis of the multiscale algorithm

As we observed above, the first (coarsest scale) stage takes time $O(N^2)$. At each successive stage, the average number of samples n_r per segment doubles, and therefore the cost of comparing two candidates increases by a factor of four. On the other hand, two segments of each candidate are compared in greater detail, which means that false (unrecognizable) candidates will be progressively eliminated.

If the elimination of false candidates happens sufficiently fast, the cost of all but the first few stages will be $O(Mn_r^2)$, where M is the number of recognizable candidates in the data. For the last (finest scale) stage, the cost is $O(Mn^2)$, where M is the number of surviving candidates. Since n_r increases in geometric progression, the total cost of the latter stages is proportional to that of the final stage. Therefore, the total cost should be $O(N^2 + Mn^2)$.

9.4 Corner blurring

When processing the coarse contours, we must take into account the fact that filtering reduces the length of recognizable candidates.

Figure 18 illustrates the problem: The ideal contours share the fracture segment $AB = CD$, but, as the corners get smoothed, the points A and B pull away from C and D, so that the recognizable candidate gets reduced to the segments UV and XY.

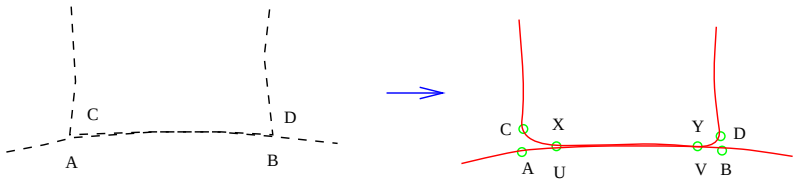


Figure 18: Blurring of fracture corners.

Because of this phenomenon, the presence of a corner at some point of the raw contour prevents the algorithm from recognizing the congruence of a few samples of the filtered contour, in the neighborhood of that point.

The number of affected samples depends on the matching threshold ξ, the filtering scale length $\bar{\lambda}$, and the sharpness of the corner. In any case, the influence of the corner decays very quickly with the distance x along the curve, roughly as $\exp(-(x/\bar{\lambda})^2)$. Therefore, the distance at which the effect is less than ξ^2 is approximately $\alpha\bar{\lambda}$, where α depends on ξ and the corner's angle. It turns out that the dependence of α on ξ is logarithmic, so in practice we can assume that α is constant. In the test example of figure 6, we have verified empirically that $\alpha = 0.6$ provides a conservative estimate of the corner effect for all scales.

9.5 Maximum filtering scale

The initial filtering scale $\bar{\lambda}_r$ that we should use is the largest one such that the desired minimum shared fracture length $C_{\min}$, minus the amount $2\alpha\bar{\lambda}_r$ of contour that is disturbed by corner blurring, is still greater than the sampling step $d_r = \bar{\lambda}_r/8$; that is

$$\bar{\lambda}_r \leq \frac{C_{\min}}{1/8 + 2\alpha} \tag{4}$$

9.6 Multiscale fragment matching algorithm

In summary, the multiscale matching algorithm can be described as follows:

Algorithm 1 *Multiscale identification of adjacent fragments*

Input:

- *the raw contours* $\{c_0, c_1, .. c_{N-1}\}$*;*
- *the initial scale length* $\bar{\lambda}_0$*;*
- *the minimum length* $C_{\min}$ *of the shared fracture line.*

Output:

- *A set* $\{W_0, W_1, .. \}$ *of plausible candidates.*

1. $r \leftarrow 0$; $d_0 \leftarrow \bar{\lambda}_0/8$;
2. While $d_r + 2\alpha\bar{\lambda}_r \leq C_{\min}$
   2.1. Filter the contours $c_0, .. c_{N-1}$ at scale $\bar{\lambda}_r$, resulting in the filtered contours $\bar{c}_0^r, .. \bar{c}_{N-1}^r$;
   2.2. $\bar{\lambda}_{r+1} \leftarrow 2\bar{\lambda}_r$; $d_{r+1} \leftarrow 2d_{r+1}$; $r \leftarrow r + 1$;
3. $r \leftarrow r - 1$;
4. Determine a set of initial candidates $\{W_0^r, .. \}$ from the contours $\{{c_0}^r, .. {c_{N-1}}^r\}$.
5. While $r > 0$
   5.1. Map the candidates $\{W_0^r, .. \}$ to the curves $\{{c_0}^{r-1}, .. {c_{N-1}}^{r-1}\}$, resulting on candidates $\left\{\hat{W}_0^{r-1}, .. \right\}$
   5.2. Combine overlapping candidates in $\left\{\hat{W}_0^{r-1}, .. \right\}$ obtaining candidates $\left\{\bar{W}_0^{r-1}, .. \right\}$.
   5.3. Refine the candidates $\left\{\bar{W}_0^{r-1}, .. \right\}$ using DP, obtaining candidates $\left\{W_0^{r-1}, .. \right\}$.
   5.4. Remove from $\left\{W_0^{r-1}, .. \right\}$ the candidates with positive mismatch; keep only the best k_r candidates per curve, and the best m_r candidates per curve pair.
   5.5. $r \leftarrow r - 1$;
6. Return candidates $\{W_0^0, .. \}$.

The limits k_r and m_r are chosen so as to balance the processing cost and the loss of true candidates. Typical values we have used are $k_r = \infty$ and $m_r = 2^r$.

9.7 Mapping candidates between scales

Recall that a candidate consists on two segments of different contours. In order to map a candidate found at some scale $\bar{\lambda}_r$ to a different scale $\bar{\lambda}_s$, we must take the time value τ_r corresponding to each endpoint of the two segments, in its respective contour c_r, and compute the corresponding time τ_s in the contour c_s, which is the same fragment as c_r filtered to the new scale.

To perform this computation, we use the reference times t_i^r and t_i^s that were produced by the filtering algorithm as a side effect of computing c_r and c_s. By definition, the points $c_r(t_i^r)$ and $c_s(t_i^s)$ are smoothed images of the same point p_i of the original contour. We locate τ_r among two consecutive times t_i^r and t_{i+1}^r, by binary search, and then compute τ_s by linear interpolation between t_i^s and t_{i+1}^s.

The times t_i^r and t_i^s are assumed to be dense enough in each curve for the interpolation error to be negligible. In any case, the result τ_s does not have to be very accurate, since the best matching for the mapped candidate will be recomputed from scratch in step 5.3, using the more detailed contours. As we discuss in a separate technical report [9], part of the task of the matching algorithm is to recompute the initial and final points of each segment. This adjustment of the endpoints is necessary in any case, to account for the blurring of fracture corners that happens at filtering as discussed in section 9.4.

Thus, after mapping a candidate from a coarse scale $\bar{\lambda}_r$ to a finer scale $\bar{\lambda}_s \leq \bar{\lambda}_r$, we must extend it slightly in both directions, so that the candidate refinement algorithm will have a chance to find the additional samples near the corners that can be matched at scale $\bar{\lambda}_s$ but not at $\bar{\lambda}_r$. More precisely, after we map a candidate from a coarse scale $\bar{\lambda}_r$ to a finer scale $\bar{\lambda}_s$, we need to extend the segments in each direction by a distance $\alpha(\bar{\lambda}_r - \bar{\lambda}_s)$; that is, by $\lceil \alpha(\bar{\lambda}_r - \bar{\lambda}_s)/d_s \rceil$ samples.

9.8 Generation of initial candidates

In order to start the multiscale process, we must solve the fragment matching problem at the coarsest scale, where the smallest shared fracture line that we wish to find has been reduced to $n_{\min} \approx 1$ sampling steps. A naive solution would be to enumerate all segments with $n_{\min}$ steps, and apply the DP algorithm [9] to every pair of such segments. If we assume that the average length of the contours is at most $C_{\min}/\beta$, for some constant β, then the average number of samples n in the coarsest contour will be $O(1/\beta)$. The cost of this solution would be $\Theta(N^2/\beta^2) = \Theta(N^2)$.

The disadvantage of this solution is that any recognizable candidate c with more than $n_{\min}$ samples will be reported as many overlapping candidates of size $n_{\min}$. Recovering the candidate c from all those little pieces is just as hard as solving the original fragment matching problem.

We could solve this problem by enumerating all pairs of samples (a_i, b_j) on all pairs of contours (a, b), and running the bidirectional DP algorithm with (a_i, b_j) as center. For each center, the DP algorithm should be applied to two segments of a and b centered at a_i and b_j and extending halfway around each contour, in each direction. Then a recognizable candidate c with $n > n_{\min}$ samples will be found approximately n times, but each of these

occurrences will be complete; so we have only to eliminate any exact duplicates from the output list. Unfortunately, each run of the DP algorithm will cost $\Theta(1/\beta^2)$, so the total cost will be $\Theta(N^2/\beta^4)$. While $1/\beta$ is a constant, it is relatively large (typically 50 to 100), so this solution is unfeasible.

Therefore, we adopt an intermediate heuristic, which assumes a $1-1$ correspondence between the samples of the two matched segments (and therefore does not need the DP algorithm), but still can find candidates longer than $n_{\min}$. The method is described below:

Recall that a contour is a circular string of samples $a = a_0, .. a_{n_a-1}$. Given two contours a and b, consider the set of pairs (i, j), where i is an index into a and j is an index into b. These pairs can be viewed as a grid Γ with toroidal topology, with periods n_a and n_b.

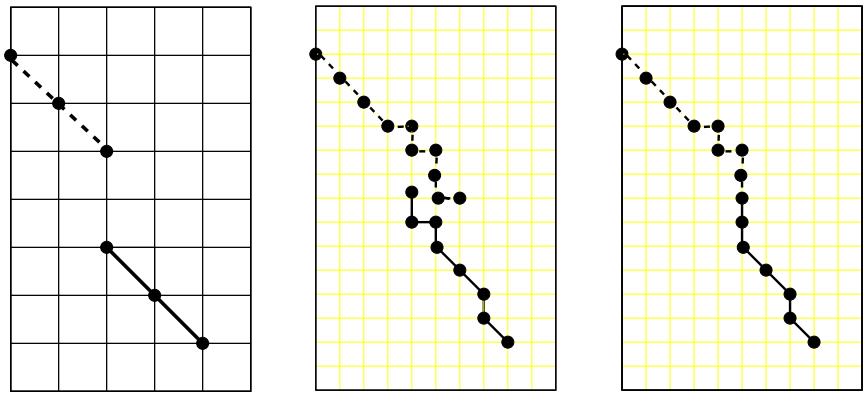
A perfect matching between a segment of a and a segment of b is a sequence of consecutive pairs (i, j) along some diagonal of this grid. Thus, we consider one diagonal of Γ at a time, and scan it looking for long sequences of index pairs (i, j) where all samples a_i and b_j are sufficiently close.

Because of the toroidal topology, each diagonal is a circular string of index pairs, that can loop several times around the torus before returning to the starting pair. The number of diagonals d of the toroidal grid is the largest common divisor of n_a and n_b. The total number of index pairs is $n_a n_b$, so the length of each diagonal is $\gamma = n_a n_b/d$, the least common multiple of n_a and n_b. We number the diagonals with indices $\{0..d-1\}$, and the elements in each diagonal with $\{0..\gamma-1\}$, in such a way that element r of diagonal number t corresponds to the sample pair (a_i, b_j) where $i = r$ and $j = (t-r) \bmod n_b$.

Each diagonal t is processed separately. For each element of the diagonal that corresponds to samples a_i and b_j, we compute the quadratic mismatch $c_r = \|a_i, bj\|^2 - \xi^2$. We set a boolean q_r to **true** whenever the sum of the $n_{\min}$ consecutive mismatches c_k starting with c_r is negative. Finally, we scan the vector q, looking for maximal substrings of **true**s; each of them is returned as an initial candidate.

9.9 Merging overlapping candidates

Since the initial candidates found in step 4 are restricted to perfect matchings, the same recognizable candidate c may still give rise to two or more partial candidates, in different diagonals, that together approximate the whole of c. Hopefully, as soon as these candidates are mapped to the next scale (step 5.1) and refined with the DP algorithm (step 5.3), they will converge to a set of overlapping pieces of c. See figure 19(b). The goal of step 5.2 is to discover and merge such overlapping partial candidates (figure 19(c)).



(a) (b) (c)

Figure 19: Merging of overlapping candidates. (a) Two initial candidates found in step 4. (b) The same candidates after refinement (step 5.3). (c) Merged candidates (5.2).

10 Results and conclusions

We now present preliminary results of our program, on two artificial but reasonably realistic test cases.

10.1 Test 1: paper fragments

The input data for this test was the set of 20 paper fragments shown in figure 20 below.

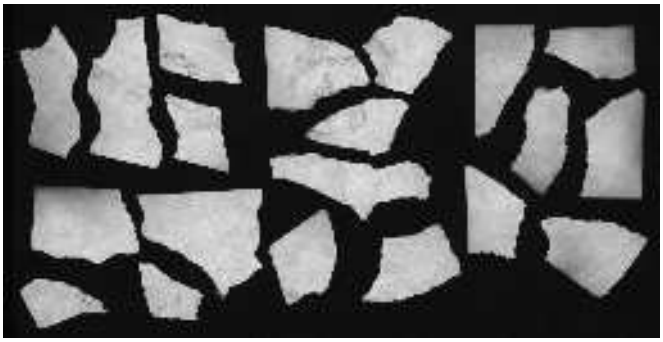


Figure 20: Input images for test 1.

These fragments were obtained from a rectangular piece of paper, about 13.0 cm by 8.0 cm, that was slightly charred to make it brittle, and then torn into pieces as shown in figure 21.

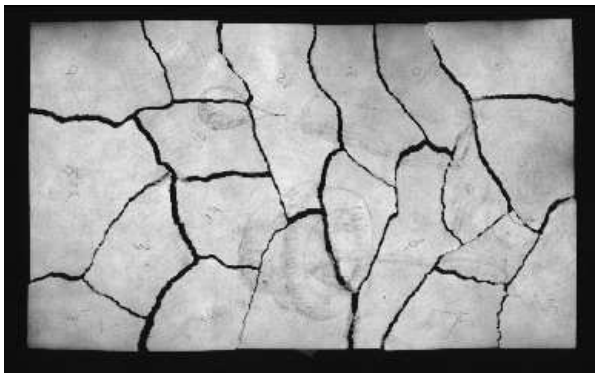


Figure 21: Test 1 fragments, manually reassembled

The 20 fragments were directly digitized with a standard flatbed scanner (UMAX model UC630 maxcolor) at 300 dots per inch, resulting in the image of figure 20. The average length L of the contours extracted from this image was 1200 pixels.

The contours were filtered and converted to coded curvature chains, as explained in section 8.2, with the following encoding parameters:

$\bar{\lambda}$ (pixels)	4	8	16	32	64	128
ϵ	0.00800	0.00600	0.00240	0.00120	0.00040	0.00020
δ	0.1892	0.1892	0.1892	0.1892	0.1892	0.1892

The multiscale matching algorithm was then applied to these coded contours, starting from scale $\bar{\lambda}_5 = 128$ pixels and ending at scale $\bar{\lambda}_0 = 4$ pixels. The target minimum segment length C_{min} was 160 pixels (1.35 cm). The table below shows the values of the threshold ξ, the maximum allowable number of candidates per pair m_r used, and the number of candidates obtained at each scale.

$\bar{\lambda}$	ξ^2	m_r	cands
128	3.0	64	5183
64	4.0	32	749
32	4.0	16	208
16	5.0	8	62
8	5.5	4	30
4	9.0	2	27

Figure 22 below shows the 27 surviving candidates (filtered at scale $\lambda = 4.0$ pixels). Each contour was rotated and translated so that the straight line segment connecting the endpoints of the matched segment was vertical and centered in the figure. The matched segments are the thicker parts of each contour (barely visible at this scale).

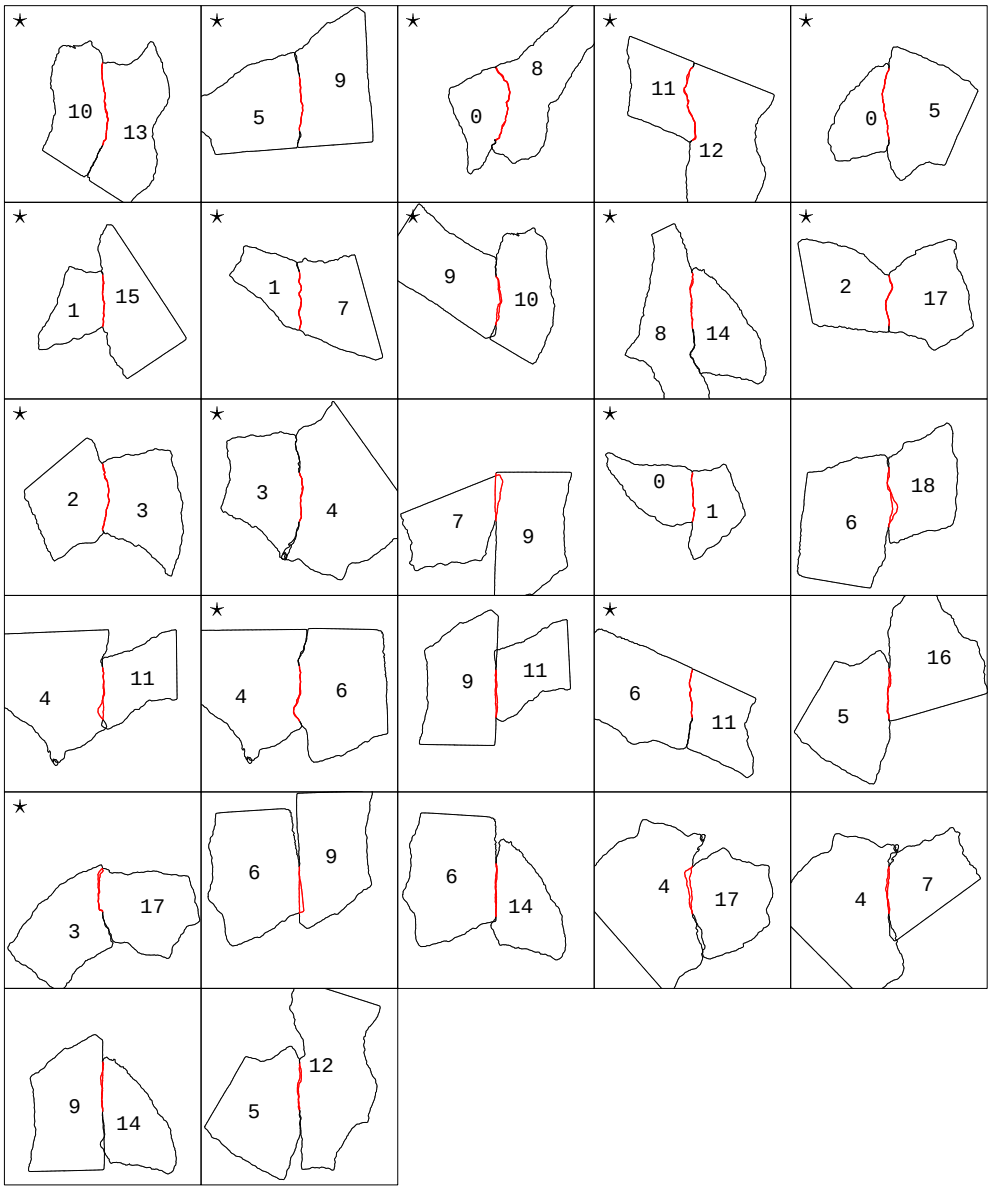


Figure 22: Candidates returned by the multiscale matching algorithm. The stars ($\star$) identify true candidates.

There were 24 recognizable candidates longer than 1.35 cm in the input data. Since 16 of the 27 returned candidates are among this set, program achieved sensitivity $\nu = 0.67$ and significance $\gamma = 0.59$ in this test.

10.2 Test 2: ceramic fragments

The input data for test 2 was the set of 112 ceramic fragments shown in figure 23 below:

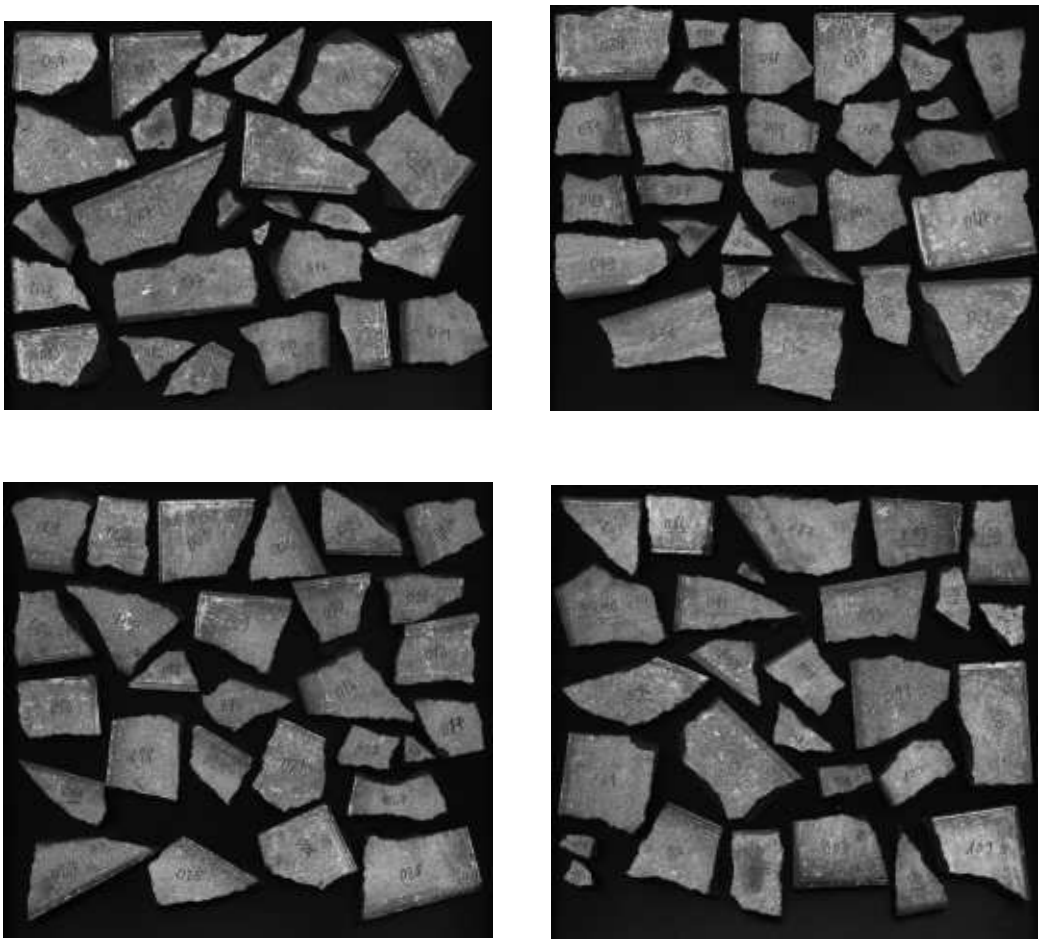


Figure 23: Input images for test 2.

These fragments were obtained by shattering five rectangular unglazed ceramic tiles, about 25.0 cm by 6.0 cm by 5mm, as shown in figure 24.

Figure 24: Test 2 fragments, manually reassembled

The fragment images, shown in figure 23, were directly digitized with a standard flatbed scanner (UMAX model UC630 maxcolor) at 300 dots per inch, after lightly rubbing the flat side of each fragment with chalk to enhance contrast. The average length L of the extracted contours was 1500 pixels.

The contours extracted from figure 23 were filtered and converted to coded curvature chains, as explained in section 8.2, with the following encoding parameters

$\bar{\lambda}$ (pixels)	8	16	32	64	128
ϵ	0.00512	0.00256	0.00128	0.00064	0.00064
δ	0.1100	0.1100	0.1100	0.1100	0.1100

The multiscale matching algorithm was then applied to these coded contours, starting from scale $\bar{\lambda}_5 = 128$ pixels and ending at scale $\bar{\lambda}_0 = 8$ pixels. The target minimum segment length C_{min} was 180 pixels (1.5 cm). The table below shows the values of the threshold ξ, the maximum allowable number of candidates per pair m_r, and the number of candidates obtained at each scale.

$\bar{\lambda}$	ξ^2	m_r	cands
128	1.50	64	85509
64	2.75	32	6896
32	3.70	16	244
16	6.00	8	77
8	8.00	4	49

Figures 25 and 26 below show the 49 surviving candidates (filtered at scale $\bar{\lambda} = 8.0$ pixels).

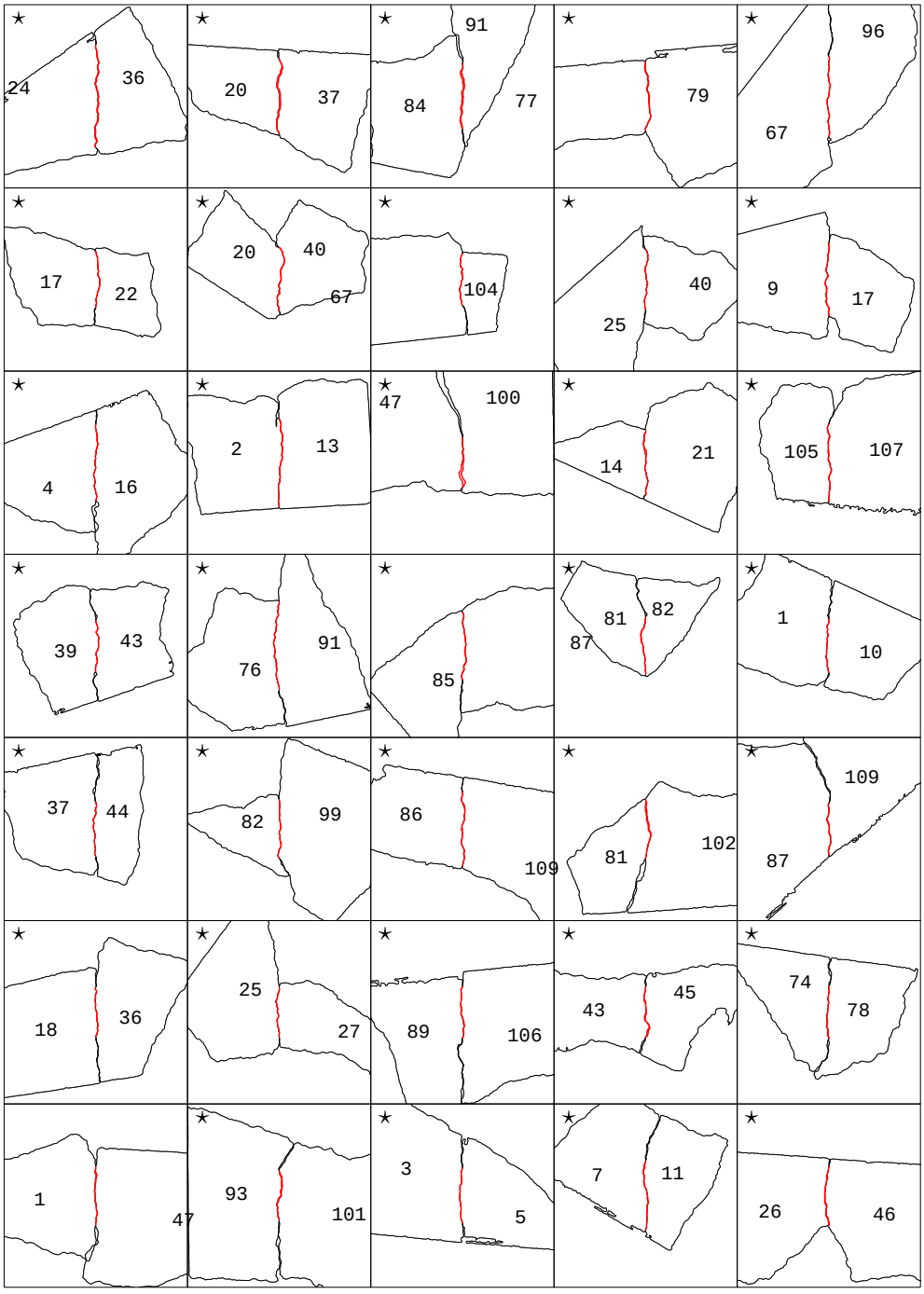


Figure 25: The best 35 candidates of the 49 returned by the multiscale matching algorithm. The stars (⋆) identify true candidates.

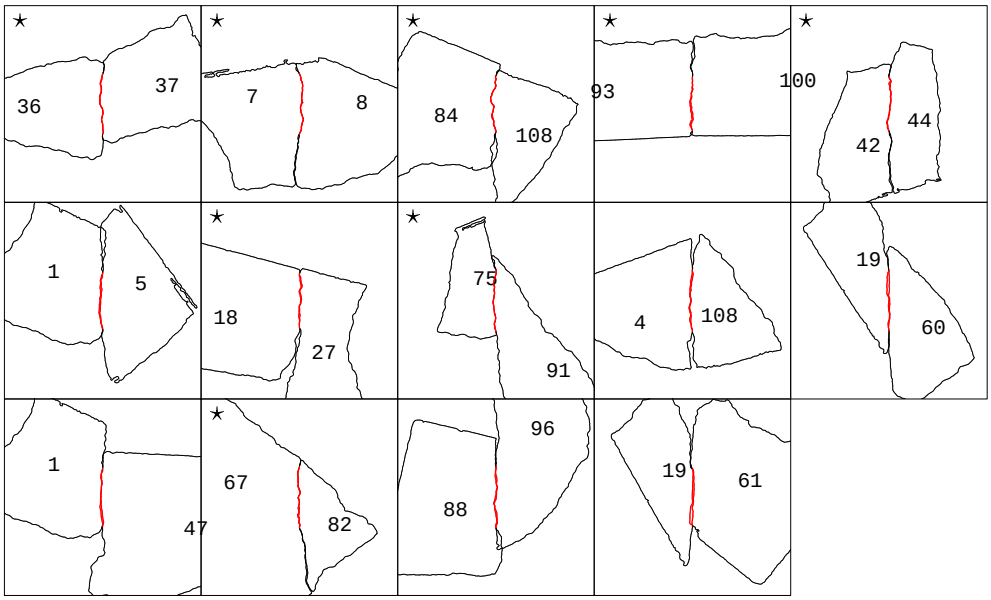


Figure 26: The next 14 of the 49 candidates returned by the multiscale matching algorithm. The stars ($\star$) identify true candidates.

There were 95 recognizable candidates longer than 1.5 cm in the input data. Since 42 of the 49 returned candidates are among this set, the program achieved sensitivity $\nu = 0.44$ and significance $\gamma = 0.86$ on this test.

10.3 Conclusions

The experiments reported above, although modest size, have demonstrated the possibility mechanically identifying adjacent fragments by matching their outlines. They also validate the basic premise of the multiscale matching method, namely that the number of surviving candidates decreases very quickly as they are re-tested with increasing resolutions.

The computational cost of our multiscale matching program is still rather large (about 2 hours for the 112 fragments of test 2). However, our implementation contains a number of known inefficiencies which can be removed with modest programming effort.

References

[1] N. Ayache and A. D. Faugeras. Hyper: A new approach for the recognition and positioning of two-dimensional objects. *Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1):44–54, 1986.

[2] J. Babaud, A. Witkin, M. Baudin, and R. Duda. Uniquess of the Gaussian kernel for scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 8(1):26–33, 1986.

[3] Roger Bagnall. Advanced papyrological information system. *File papyrus/texts/APISgrant.html at http://scriptorium.lib.duke.edu/*, 1994.

[4] H. Bartels J. C. Beatty and B. A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers, 1987.

[5] F. Boussofiane and G. Bertrand. A new method for recognizing and locating objects by searching longest paths. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 26(12):445–448, 1993.

[6] Ronald N. Bracewell. *The Fourier Transform and its Applications.* McGraw-Hill, 1986.

[7] Grigore C. Burdea and Haim J. Wolfson. Solving jigsaw puzzles by a robot. *IEEE Transactions on Robotics and Automation*, 5(6):752–764, 1989.

[8] Roland T. Chin and Charles R. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys*, 18(1):67–108, 86.

[9] Helena Cristina da Gama Leitão and Jorge Stolfi. Comparing fracture lines. Technical report - in preparation, Institute of Computing(IC), University of Campinas, 1998.

[10] Helena Cristina da Gama Leitão and Jorge Stolfi. Geometric curve filtering. Technical report - in preparation, Institute of Computing(IC), University of Campinas, 1998.

[11] Alan D. Kalvin et al. Using visualization in the archaeological excavations of a pre-Inca temple in Peru. Report RC 20518, IBM T. J. Watson Research Center,, 1996.

[12] K. Falconer. *Fractal Geometry : Mathematical Foundations and Applications.* John Wiley & Sons, 1990.

[13] K. T. Glowacki. Franchthi excavations: 17,000 years of Greek prehistory. Dept. of Classical Studies, Indiana University, Bloomington. *http://www.indiana.edu/ archaeol/franchthi/franchthi.html*, 1990.

[14] Multimedia Hochfeiler. Gli archivi di Ebla. *http://www.sysin.it/ipertest/archi.htm*, 1997.

[15] Alan Kalvin, Edith Schonberg, Jacob T. Schwartz, and Micha Sharir. Two-dimensional, model-based, boundary matching using footprints. *The International Journal of Robotics Research*, 5(4):38–55, 1986.

[16] Joan Keller. The glass from Sepphoris: A preliminary report. *http://www.colby.edu/rel/Glass.html*, 1995.

[17] Raymond Legault and Ching Y. Suen. Optimal local weighted averaging methods in contour smoothing. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 19(8):801–817, 1997.

[18] James W. McKee and J. K. Aggarwal. Computer recognition of partial views of curved objects. *IEEE Transactions on Computers*, c26(8):790–800, 1977.

[19] João Meidanis and João Carlos Setubal. *Introduction to Computational Molecular Biology.* PWS, 1997.

[20] Farzin Mokhtarian and Alan Mackworth. Scale-based description and recognition of planar curves and two-dimension shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 8(1):34–43, 1986.

[21] Farzin Mokhtarian and Alan K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 14(8):789–805, 1992.

[22] Arthur R. Pope. Model based object recognition: A survey of recent research. Technical Report TR-94-04, Berkeley University, California, 1994.

[23] Azriel Rosenfeld and Avinash C. Kac. *Digital Picture Processing.* Academic Press, 1982.

[24] Paul Rosin and Svetha Venkatesh. Extracting natural scales using the Fourier description. *Pattern Recogniton*, 26(9):1383–1393, 1993.

[25] B. Sandak, R. Nussinov, and Haim J. Wolfson. An automated computer vision and robotics based technique for 3-d flexible biomolecular docking and matching. *Computer Applications in the Biosciences*, 11(1):87, 95.

[26] Behzad Shahraray and David J. Anderson. Optimal estimation of contour properties by cross-valided regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 8(6):600–610, 1989.

[27] Fred J. Taylor. *Digital Filter Design Handbook (Eletrical Engineering and Eletronics 18).* Marcel Dekker, 1983.

[28] Göktürk Üçoluk and I. Hakki Toroslu. Reconstruction of 3-d surface object from its pieces. In *The Ninth Canadian Conference on Computational Geometry - Queen's University, Kingston, Ontario, CANADA, August 11-14*, volume 1, 1997.

[29] R. Vergnieux. Le fac-similé èlectronique ou les restitutions virtuelles, - X^e CNRS Table Ronde de Informatique et Égyptologie, Bordeaux. *http://silicon.Montaigne.u-bordeaux.fr/8001/HTML/ONLINE/IE10/Robert.html*, 1994.

[30] Andrew P. Witkin. Scale-space filtering. In *Proc. IJCAI Eighth Proc. International Joint Conference on Artificial Intelligence - Karlsruhe - Germany*, pages 1019–1021, 1983.

[31] Haim J. Wolfson. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 12(5):483–488, 1990.

[32] Daniel M. Wuescher and Kim L. Boyer. Robust contour decomposition using a constant curvature criterion. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 13(1):41–51, 1991.