COMPUTER-AIDED DESIGN & APPLICATIONS, 2017
VOL. 14, NO. 6, 796–804
https://doi.org/10.1080/16864360.2017.1287755



# On the nesting of a contour dataset and its use in repair

John K. Johnstone

University of Alabama at Birmingham, USA

**ABSTRACT**

A contour dataset is a common source of data in biomedical imaging and GIS. In practice, many contour datasets violate key assumptions: contours self-intersect or intersect other contours, which can disrupt the performance of algorithms. We call such data dirty. Avoiding this issue, algorithms in the literature assume clean data. This paper considers the issue of dirty datasets, in the context of nesting.

Contours are often nested inside other contours, and the nesting level of a contour is important, as it affects the position of the inside of the shape. This paper analyzes the nesting of a dirty dataset, then shows how to use this nesting analysis to repair the data. A theme is the power of image space algorithms in the presence of noise. Image space algorithms for polygon area and Boolean operations of polygons are provided.

**KEYWORDS**

Nesting; repair of a dataset; contour reconstruction; biomedical modeling

## 1. Introduction

A contour dataset is a common source of data in biomedical imaging and GIS. It can arise from a CT/MR scan that models anatomy (Fig. 1) or from a topographic map that models terrain. In practice, a contour dataset is often imperfect, violating key assumptions. Such dirty data causes problems downstream in working with the dataset, and we would like to repair the data. We explore this issue through the lens of nesting, an important property of a contour dataset. We first show how to analyze nesting in a dirty dataset. We then show how to use this nesting analysis to repair the dataset.

The rest of the paper is structured as follows. The rest of this section introduces the issue of nesting and dirty datasets, including the importance of nesting to contour reconstruction. Section 2 considers a naive algorithm to compute nesting level, and how it breaks down with dirty datasets, then gives our robust algorithm to compute nesting level and nesting parent. Section 3 discusses the image space algorithms for area that this nesting algorithm requires, then shows how to repair a dirty dataset using the nesting analysis and the image space algorithms. Section 4 discusses the complexity of our algorithm, evaluates the algorithm against a collection of datasets, including timing and the preponderance of nested and dirty datasets, and discusses how to encode nesting in a dataset. We end with some conclusions and ideas for future work.

### 1.1. Contours and their nesting

**Definition 1.1.** *A curve or surface is simple if it does not self-intersect. A Jordan curve is a simple closed curve in the plane. The interior of a Jordan curve is well defined, by the Jordan curve theorem. A contour is a point sequence defining a Jordan curve. A contour dataset is a point cloud arising from the slicing of a set of simple orientable 2-manifolds without boundary by a series of parallel planes. It is composed of contours.*

In principle, two contours C and D from the same slice have only three possible relationships: the interiors of C and D are disjoint, C lies completely inside D, or D lies completely inside C (since a contour dataset is sampled from simple 2-manifolds). Thus, some contours are nested inside other contours.

**Definition 1.2.** *The nesting level of a contour C is the number of contours in the same slice that contain C. A contour is nested if it has a positive nesting level. A nested contour, say of level n, has a nesting parent: the unique contour of nesting level n-1 in the same slice that contains C. A contour dataset is nested if it contains a nested contour.*

**Example 1.3.** *Consider Fig. 2. In Fig. 2a, there is one contour C of nesting level 0 and three contours of nesting level 1, all of which have C as their nesting parent. In Fig. 2b, there are six nested contours, all of level 1. In Fig. 2c, there*

**CONTACT** John K. Johnstone jkj@uab.edu

**Figure 1.** Contour dataset of a jaw. Nested contours are highlighted.

*are six unnested contours of level 0, many nested contours of level 1, and three nested contours of level 2.*

The nesting behaviour of a contour dataset is important, because it defines the inside of the shape defined by the dataset. Contours at even nesting levels represent the shape of interest, while contours at odd nesting levels represent holes in this shape. Since nesting captures the position of the inside of a shape relative to a contour, the nesting level of a contour dictates its treatment. For example, during reconstruction, contours at odd nesting levels should be connected only to other contours at odd nesting levels.

Most interesting datasets are nested. 23 of the 29 contour datasets in Barequet's repository [1] are nested, and 9 of the 12 in Geiger's repository [6]. Datasets with a high percentage of nested contours are also common, especially in biomedical datasets. In one of Barequet's mandible datasets, 4688 of the 5012 contours are nested.

As the nesting behaviour of a contour dataset affects most downstream analysis of that contour dataset, it is important to analyze nesting early in the processing of a dataset, essentially the first analysis after reading. The challenge, as we shall see in the next section, is to make the nesting analysis robust to dirty datasets.

**Definition 1.4.** *A nesting analysis of a contour dataset is a computation of the nesting level of every contour, and the nesting parent of every nested contour.*

This paper will develop a robust algorithm for nesting analysis. The complicating factor is that contours are often dirty.

### 1.2. Dirty contours

A contour is clean if it is free from all types of intersection. In principle, all contours of a contour dataset are clean, since a contour dataset is sampled from simple 2-manifolds. However, although in theory the boundary of a contour should not self-intersect, in practice it often does; and although in theory two contours are either disjoint or one is nested entirely inside the other, in practice this relationship may be violated slightly and the contours intersect. We can view these as noisy versions of the Platonic ideal (Fig. 3).

**Definition 1.5.** *A contour is clean if it is simple and its interior does not intersect any other contour's interior. A contour is dirty if it is not clean (not simple or its interior intersects the interior of another contour). A contour dataset is clean (resp., dirty) if none (resp., any) of its contours are dirty.*

Dirty datasets are common. Natural sources of dirty data are noise in image acquisition and noise in image segmentation. For example, most of the contour datasets we received from a major biomedical company were dirty, much of the topographic data from the USGS is dirty, and a slicing of the biomedical meshes from some other research groups reveals that they are sometimes self-intersecting, indicating that they probably arose from dirty contour datasets.

**Figure 2.** A single slice of a contour dataset typically contains some nested contours and some unnested contours. (a) A pelvis slice. (b) A brain slice. (c) A skull slice.

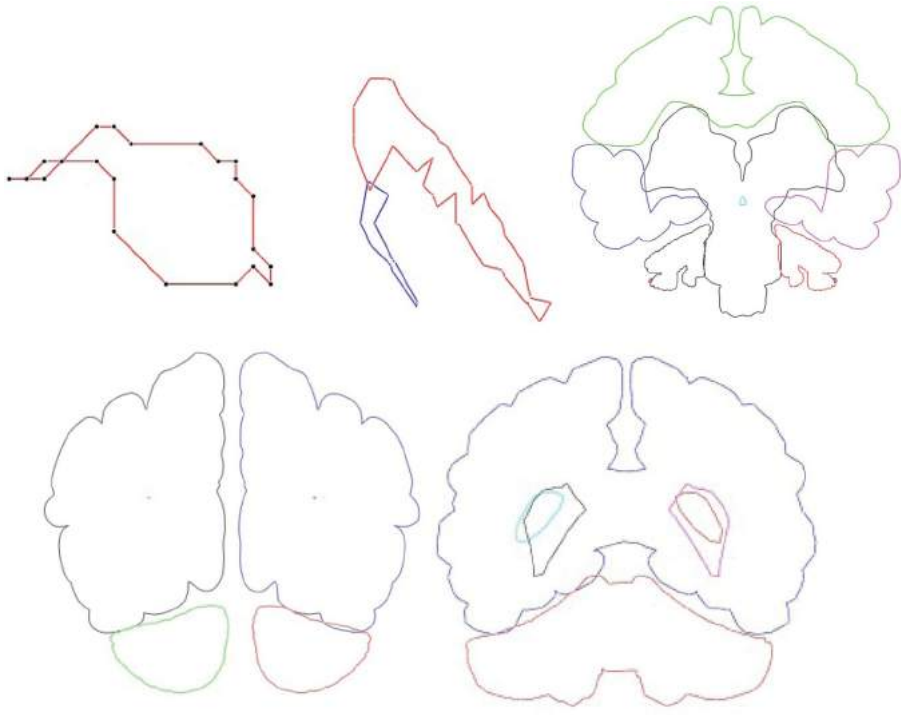

**Figure 3.** Dirty contours. (a) A non-simple contour from a jaw dataset. (b) Intersecting contours from a USGS topographic map. (c-e) Intersecting contours from a brain dataset.

Unfortunately, the contour reconstruction literature uniformly assumes clean contour datasets. A few examples will illustrate. Barequet and Sharir [2] assume the contour dataset is 'a series of parallel planar slices . . . each consisting of a collection of *non-crossing*, but possibly nested, closed and *simple* polygonal curves' (our emphasis) and in a later paper [3], 'a set of n consecutive slices . . . , each composed of closed simple contours (that can be nested but otherwise not intersecting)'. Geiger [7] assumes 'the object contours are given as a set of straight line segments, forming one or several simple closed polygons'. Ju et. al. [8] assume 'a set of non-intersecting curve networks'. In other words, the algorithms in the contour reconstruction literature depend on clean data for correct behaviour. Although it is likely that all of these algorithms do some cleanup to ensure clean datasets, none is discussed. The lack of treatment of the issue of dirty datasets, despite their prevalence, was the inspiration for this paper.

The major goal of this paper is nesting analysis, even of dirty datasets. We will also show how this nesting analysis can be used to clean up a dirty dataset. A theme of the paper will be the success of image space algorithms in handling dirty data.

### *1.3. The importance of nesting analysis*

To illustrate the importance of nesting in contour reconstruction, consider the reconstruction of two neighbouring sections, first in an unnested dataset (Fig 4a) and then in a nested dataset (Fig. 4d) of a torus. In both cases, one section contains a single contour and the other section contains two contours (Fig. 4c,f). When a contour branches to two contours C1 / C2 on the next section, and none of the contours are nested, the correct reconstruction has the contours C1 / C2 merging (Fig. 4c). However, when a contour branches to two contours C1 / C2 on the next section, and C1 is nested inside C2, the correct reconstruction has the singleton contour merging with itself (Fig. 4f). This example shows how nesting can affect the behaviour of a reconstruction algorithm.

Another example of the importance of nesting information is in the popular strategy of computing contour topology using overlap. Nesting needs to be taken into account when computing overlap, since a contour of odd nesting level represents a hole (its interior is empty). For example, in Fig. 4f, the region inside the contour of nesting level 1 is empty.

## 2. Nesting analysis

If datasets were always clean, the computation of nesting level would be straightforward. In a clean dataset, two contours either have no intersection or one is contained in the other. Therefore, if a pair of contours on the same slice overlap, the nesting level of the contour of smaller

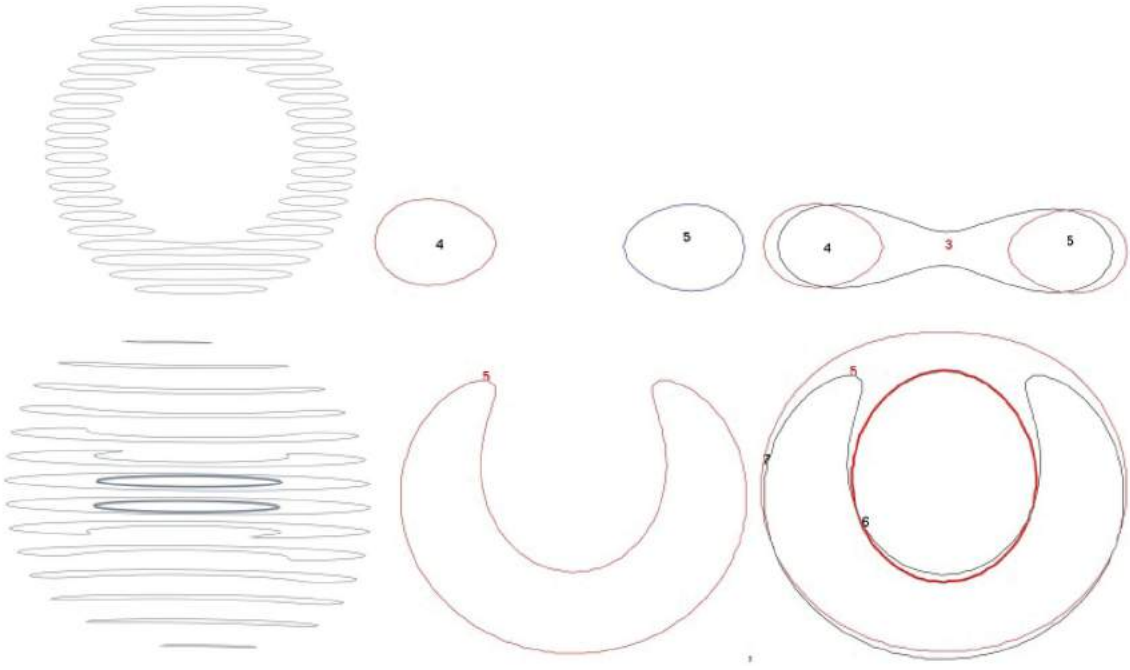


**Figure 4.** Nesting affects reconstruction. Consider an unnested dataset of a torus (a). If unnested contours C1 / C2 on section i (b) branch to a contour D on section i+1 (c), then C1 and C2 should merge. On the other hand, consider a nested dataset of a torus, using different slicing planes (d). If two nested contours C1 / C2 on section i branch to a contour D on section i+1 (e-f), then D should merge with itself.

area should be incremented. This suggests an algorithm using intersection.

**initialize the nesting level of all contours to 0**
**compute the area of all contours**
**for each pair of contours C, D in the same slice,**
**if C intersects D, increment the nesting level of the contour of smaller area**

However, this algorithm is brittle to the types of noise present in dirty datasets: if two contours at the same nesting level slightly overlap, they will be assigned the wrong level. Also note that object space algorithms for intersection are difficult and are designed for simple polygons [9]. Consider another algorithm that uses point location.

**initialize the nesting level of all contours to 0**
**for each pair of contours C, D in the same slice,**
**choose an arbitrary point c inside C and an arbitrary point d inside D**
**if c lies inside D, increment the nesting level of C**
**else if d lies inside C, increment the nesting level of D**

This algorithm is also brittle to dirty datasets: if two contours at the same nesting level slightly overlap and a point in the overlap is chosen as the candidate for point location, the two contours will be assigned different nesting levels. Another problem is the challenge of developing a robust implementation of point location [4]. If a contour is not simple (or almost not simple), problems may arise with point location against this contour [5]. We see that the challenge is a robust implementation of nesting analysis in the context of dirty datasets.

Now consider our algorithm for nesting analysis, robust to dirty datasets. Its preprocessing phase is:

**for each contour C,**
**compute its area a(C)**
**for each pair of contours C, D in the same slice,**
**compute their area of intersection a(C,D)**

followed by the nesting level phase of nesting analysis:

**initialize the nesting level of all contours to 0**
**for each pair of contours C, D in the same slice,**
**if a(C,D) > .5 * min (a(C), a(D))**
**increment the nesting level of the smaller contour**

This algorithm uses area of overlap as the dominant diagnostic tool: if two contours overlap by more than half the area of the smaller contour, they have a nesting relationship. Why half the area? To explore this constraint, first note that there is a limit to the amount of noise that an algorithm can withstand. If two unnested contours

mistakenly intersect by a little bit, we can recognize this error and correct it by pulling the contours apart so they have no overlap. On the other hand, if a nested contour mistakenly intersects its parent contour by a little bit, we can recognize this error and correct it by pulling the child inside the parent so they overlap completely. But to distinguish these two cases, we must assume that if a contour intersects more than half of another, it is nested within it; while if a contour intersects less than half of another, it is not nested within it. Otherwise, there is no way of distinguishing nested contours from unnested contours. This sets the limit of mistaken overlap to half the size of the smaller contour.

In theory, the intersection of two contours should be either empty or the size of the smaller contour; therefore, in practice, it is either close to 0 or close to the size of the smaller contour. Since we are comparing to half the area of the smaller contour, this algorithm yields accurate results even in the presence of significant noise (overlap of up to half the area of the smaller contour, violations of simplicity). Fortunately, the errors in a dirty dataset are usually limited: a contour will be close to simple, or two contours will overlap slightly.

The algorithm above is robust to both types of noise in a dirty contour: two contours at the same nesting level may overlap slightly, and a child need not fully lie inside its parent. More subtly, a contour need not be simple, as long as we can compute its area. Another way in which the algorithm is robust to noise is that computing the exact area of a contour is not important, only its relative size. The precise intersection of two contours is also not important, just its relative size. Therefore, both calculations are robust to noise.

An image space algorithm for area of a polygon is used, one that is robust to non-simple polygons (see the next section). The intersection of two polygons can also be computed in image space, again a more robust and simple computation.

A robust algorithm to compute nesting parent is as follows (the nesting parent phase of nesting analysis):

```
compute the nesting level of all contours
for each contour C of positive nesting level n
    for all contours D in C's slice of nesting level n-1
        if a(C,D) > .5 * min(a(C), a(D))
            D is the nesting parent of C
```

That is, the nesting parent D of a contour C of level $n > 0$ is the unique contour in C's slice of nesting level n-1 that intersects C nontrivially. If there are two or more contours with this nontrivial overlap with C, the contour dataset is too dirty for our algorithm. This can be used to evaluate when a dataset cannot be handled, so that the algorithm fails robustly rather than computing a false answer. Notice again that exact areas are not important, and that area must be computed robustly in the presence of dirty contours.

We next look at how image space algorithms can be of help in implementing this algorithm, both for robustness and simplicity.

## 3. Image space algorithms and their use in repair

This section first develops robust image space algorithms to compute the area of a polygon and the area of intersection of two polygons, even if the polygons are dirty. Not only are these algorithms fundamental to the nesting analysis, but they can also be used for the repair of a dataset, which is addressed at the end of this section. Since exact areas are not needed for nesting analysis, only relative areas, it is fine that these areas are measured in pixels in image space. However, if exact areas are wanted, they can be computed easily: if the scaling factor used in mapping to the unit cube (see the next paragraph) is s, multiply the calculated areas by $s^2$.

### 3.1. Image space algorithms for area

First a preprocessing step. In these algorithms, the polygon or pair of polygons must be entirely visible to the camera, and orthogonal to the camera so that rendering does not change its area. To accomplish this, the scene is rendered by a camera that looks down the z-axis at the unit cube, using orthographic projection. The contour dataset is rotated so that the normal of the contour planes agrees with the z-axis, and translated/scaled so that the entire contour dataset fits in the unit cube. This guarantees that each contour fits in the frame buffer and its area is not warped. Also note that, since our analysis of nesting requires the frame buffer, it should be done independently of downstream analysis.

The area of a polygon is computed in image space, as follows: clear the color buffer to black, fill the polygon in red, and count the red pixels. The area of the intersection of two polygons is computed in image space by clearing the color buffer to black, filling the first polygon in red, filling the second polygon in blue using addition blending, and counting the purple pixels (Fig. 5a). To count purple pixels, read the pixels with any red component (say using OpenGL's glReadPixels), read the pixels with any blue component, and count the pixels that are both red and blue. A polygon is filled in image space, as follows. The filling algorithm is provided for completeness, as filling a polygon is well understood. The color buffer is cleared and disabled, the stencil buffer is cleared and

enabled, and the draw mode is set to fill. The polygon P is drawn into the stencil as a triangle fan (as if it were convex), inverting the stencil with each fragment render. Pixels inside the polygon are associated with pixels whose lowest order stencil bit is 1, since pixels inside the polygon are covered by an odd number of triangles and so have an odd number in the stencil. The color buffer is enabled and the polygon P is redrawn (again as if it were convex) but only where the stencil is odd. The color buffer now contains the filled polygon. Although this algorithm is designed for a simple polygon, it fails gracefully if the polygon is almost simple (Fig. 5b,c).

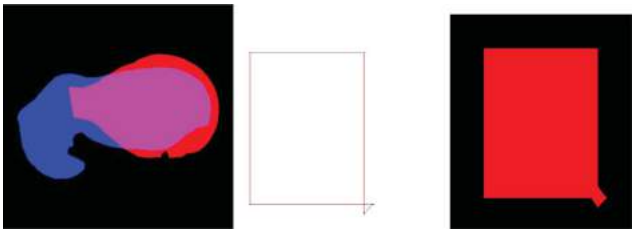

**Figure 5.** Filling. (a) Boolean operations on two polygons may be computed in image space by filling one polygon in red and the other polygon in blue: the intersection is now purple, the two differences are red and blue, and the union is red, blue or purple. (b-c) The filling of a nonsimple polygon repairs it.

Note that our construction for the intersection of two polygons actually computes all Boolean operations of the polygons, not just intersection (Fig. 5a). The intersection is purple, the union is red, blue or purple, one difference is red, and the other difference is blue. This availability of all Boolean operations will be useful during repair. Also note that these image space algorithms are oblivious to contour orientation (unlike some object space algorithms), which is a good thing since orientation may be incorrect in dirty data.

### 3.2. The repair of intersecting contour pairs

The nesting analysis of a dataset that we have just introduced (Sections 2 and 3.1), which works even on dirty datasets, may be turned around and used to guide a repair of the dataset into a clean dataset (Defn. 1.5). It turns out that the repair may be implemented using the same image space algorithms developed in Section 3.1. Although repair could be approached in other ways, the novelty of the proposed algorithm is that the repair is incorporated into the same infrastructure as the nesting analysis, using the same image space algorithms for area and intersection and using the nesting analysis to diagnose and guide the repair.

Consider two intersecting contours (see Fig. 3e). A repair strategy is to remove the intersection by pulling the contours apart. Since repair must preserve the nesting levels, nesting gives a hint to the direction of motion of the contours during repair and which contours should change. If two contours of the same nesting level intersect (Fig. 3d), they should be contracted away from each other, and each move towards its inside. But if a child, say of level i+1, intersects its parent, say of level i (Fig. 3e), the child should be contracted away from the parent, towards the child's inside, but the parent should not contract towards its inside, since this motion of the parent would only increase the intersection with the child. The parent could move outwards, but we decide to keep it stationary.

An interesting case to consider is the case when three or more contours intersect, all at different levels. Consider a contour C of level 2 inside a contour B of level 1, both of which intersect their parent (grandparent) A of level 0. Our principle will be to repair two contours of neighbouring levels at a time, moving down through the levels. In other words, the contour of level 1 first moves inside the contour of level 0. Then the contour of level 2 moves within the contour of level 1. Note that this also repairs its intersection with the contour of level 0. This restriction to neighbouring levels limits the number of contour pairs that need to be considered.

Now that we know which contours to change, consider how to repair the contour. There are many strategies to consider: translation of the entire contour, motion of a local subset of points near the overlap, or removal of the intersection from both contours. We use the last option: simply removing the violating intersection, since the other repair options can have a ripple effect and create other violating intersections.

Consider two intersecting contours $c_1$ and $c_2$ of nesting levels $n_1$ and $n_2$. Since we only repair contours at the same level or neighbouring levels, we may assume that $n_1 = n_2$ (same level) or $n_1 = n_2 + 1$ ($c_1$is nested inside $c_2$). If $n_1 = n_2$, $c_1$ is set to $c_1 - c_2$ and $c_2$ is set to $c_2 - c_1$, both of which remove the intersection. Recall that the difference of two contours can be computed using our image space algorithm (Fig. 5a). The image of the difference $c_1 - c_2$ or $c_2 - c_1$ can be converted into a polygon by tracing its boundary (we suggest the use of the 'findContours' operator in OpenCV). Now consider the other case. If $n_1 = n_2 + 1$, $c_1$ is set to $c_1 \cap c_2$ (moving the child $c_1$ inside the parent $c_2$). As with the $n_1 = n_2$ case, the image of the intersection $c_1 \cap c_2$ is converted into a polygon by tracing its boundary. Finally, we must lift the repaired 2D contours into 3-space, by giving them a z-coordinate: the ratio of the perimeter of the original contour to the perimeter of the new image space contour gives the scaling, which is used to reset the original distance between

sections, which defines the z-coordinate. In conclusion, we have the following repair algorithm.

**for each slice**
  **for each nesting level i in this slice**
    **for each pair of intersecting contours $c_1$, $c_2$ of level i**
      **$c_1$ is replaced by $c_1 - c_2$ and $c_2$ is replaced by $c_2 - c_1$**
    **for each pair of intersecting contours $c_1$ of level i+1 and $c_2$ of level i**
      **$c_1$ is replaced by $c_1 \cap c_2$**

Note that the intersecting contours were already detected during the preprocessing phase of nesting analysis (Section 2): they are the contour pairs whose area of intersection is positive. Finally, we observe that if a single dirty contour is discovered, the entire dataset is replaced, moving it to image space: since one of the contours must be replaced by its image space version, all must be replaced for consistency. Note that this makes repair an expensive operation.

### 3.3. The repair of a nonsimple contour

An image space algorithm can also be used to repair nonsimple contours. When a nonsimple polygon is filled in image space, the resulting shape has a simple boundary (Fig. 5b-c). Therefore, a nonsimple contour can be repaired by replacing it by its boundary when it is filled in image space.

We end this discussion of repair by noting its weaknesses. We have already noted its expense. Since image space is used, the speed of the repair depends on the quality of the GPU, and the quality of the repair is bounded by the image space resolution. For example, very small contours can be lost during repair in huge datasets. Accuracy could be improved by dividing the dataset into subsets, effectively increasing the resolution. Another weakness is that there is a limit to the size of error that can be corrected. For example, if a contour's topology is changed by its self-intersection (say adding a hole), this mistake in topology cannot be repaired by our algorithm. Fortunately, most segmentation errors are small and local in biomedical datasets. Another approach is to avoid repair by developing algorithms that are robust to imperfect data, like our nesting analysis.

## 4. Results and validation

We now report some results, based on our C++ implementation of these algorithms. We concentrate on nesting analysis, since that is the main result of this paper.

First consider the complexity of our nesting analysis (Section 2). Let $n_s$ be the number of contours in slice $s$, $c_a$ be the cost of a polygon area computation, and $c_i$ be the cost of an intersection area computation. Since a nesting analysis requires computation of the area of all contours that lie in slices that contain more than one contour, and the area of intersection of all contour pairs that lie in the same slice, the cost of the nesting analysis of a contour dataset is:

$$\sum_{s,n_s \geq 2} n_s * c_a + \binom{n_s}{2} * c_i$$

Of the 52 contour datasets we analyzed, 38 (73%) are nested, more than 10% of the contours are nested in over half of the datasets (29), 27 datasets have more than 10 nested contours, 17 (33%) have more than 100, and 7 datasets (13%) have more than 1000 nested contours. Some datasets have a huge number of nested contours: for example, a mandible dataset from Barequet's repository has 5012 contours, 4657 of which are nested. In short, contour datasets are usually nested and some datasets are very nested. Fig. 6 shows examples of nesting as computed by our code.

The median time required for a nesting analysis is 9.5 seconds. The mean time for a nesting analysis is 1.5 seconds for datasets with up to 100 contours, and 9.5 seconds for datasets up to 1000 contours. Since nesting analysis is inherently quadratic, it can be expensive for huge datasets: an extreme case would be the 5012 contours of the mandible dataset, which required almost 15 minutes. Since this is an image space algorithm, the limiting factor is the quality of the GPU (our results are based on a 4 year old NVIDIA GeForce GTX 675MX).

26 of the 52 datasets are dirty: 24 datasets (46%) have contours that are not simple, and 7 datasets (13%) have intersecting contours. An extreme case is a kidney dataset with 4160 contours, 1472 of which violate the simplicity constraint; and a brain dataset with 44 contours, 25 of which intersect. The median time for a cleanliness analysis (finding dirty contours) is 2.1 seconds. Huge datasets are again expensive, because of the quadratic nature of testing for intersections, which is done in image space. The 5012 contours of a mandible dataset took 8.3 minutes. As we have noted, repair of a dirty dataset is also expensive: for example, repair of a typical dataset took over 8 minutes. Fig. 7 shows an example of the repair of a brain dataset. Fortunately, nesting and cleanliness analysis are preprocessing steps that can be parallelized and decoupled from later analysis.

The nesting level and nesting parent can easily be encoded in a contour dataset. In our format, a contour of nesting level 0 is recorded as 'C < indices of points in contour >', while a contour of positive nesting level n

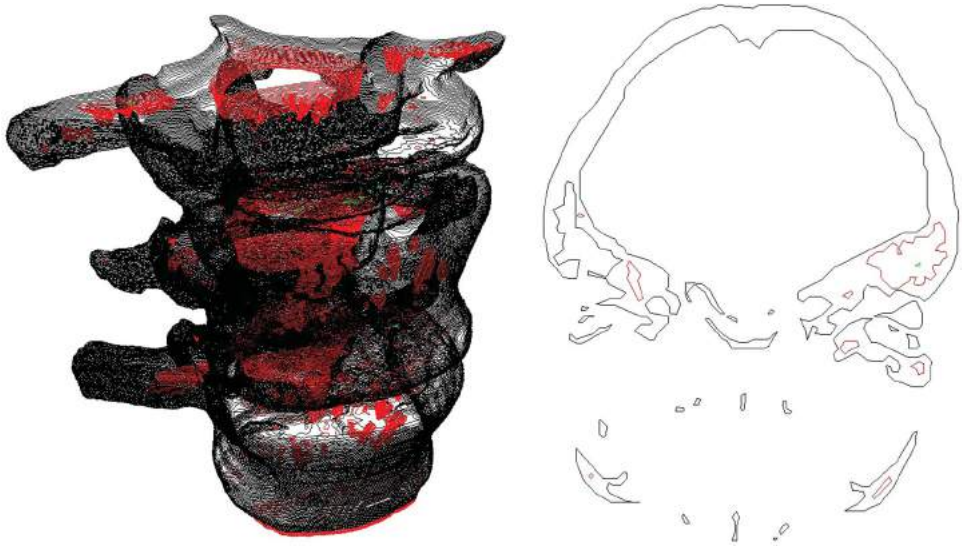

**Figure 6.** (a) The nesting of a spine dataset: red contours are nested. (b) The nesting of a skull dataset, as computed by our implementation, showing one section colour coded by nesting level.

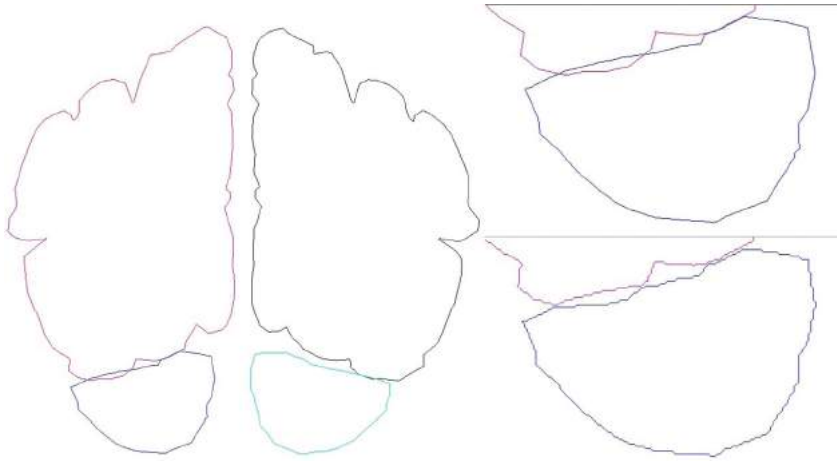

**Figure 7.** Intersecting contours (left), and zoomed in on one of the intersecting regions (top right). The repaired contours, as computed by our implementation, are shown on bottom right. At the small scale of this repair, note the pixelation of the contours due to the image space nature of the repair.

with nesting parent p is recorded as 'CN n p < indices of points in contour > '.

Once nesting has been analyzed, contours should be oriented consistently with their nesting level: contours of even nesting level should have counterclockwise orientation, while contours of odd nesting level should have clockwise orientation. The inside of the shape is now consistently to the left, and has been encoded into the contour dataset.

## 5. Conclusions and future work

A robust algorithm to analyze the nesting of a contour dataset has been developed, even if the dataset is dirty. Since noise in data is common, it is important to develop algorithms that handle noise. Most nontrivial datasets are nested and knowledge of nesting is important for their reconstruction. A lesson learned is that an analysis of nesting benefits from the use of area rather than point location, and image space rather than object space. Area in image space is an analog computation, forgiving of error. Even though the use of image space algorithms was motivated by the need for robustness, these algorithms are also simpler to implement than their object space counterparts. We conclude that, by using components that fail gracefully on dirty datasets, embedded in an algorithm that only requires approximate answers, a perfect analysis of nesting is possible even for an imperfect dirty dataset. The nesting analysis can then be used to repair the dataset.

In future, we want to use our knowledge of nesting directly in contour reconstruction. We also want to apply our nesting analysis to computation of the topology of a contour dataset (how contours connect between different sections): nesting is an important clue since nesting encodes holes. Finally, we want to explore the interaction between the nesting of contours within a slice and the branching of contours between slices, which exhibit some duality.

## Acknowledgements

I am grateful to the community of scholars who have made contour datasets available, including the biomedical datasets of Bajaj, Barequet, Columbia, Geiger and the topographic maps from the USGS. I also thank the reviewers for their guidance.

## ORCID

*John K. Johnstone* http://orcid.org/0000-0003-4033-0066

## References

[1] Barequet, G., ftp://ftp.cs.technion.ac.il/pub/barequet/psdb, contour dataset repository.

[2] Barequet, G.; Sharir, M.: Piecewise-Linear Interpolation between Polygonal Slices. Computer Vision and Image Understanding, 63(2), 1996, 251–272. http://dx.doi.org/10.1006/cviu.1996.0018

[3] Barequet, G.; Vaxman, A.: Nonlinear Interpolation between Slices, International Journal of Shape Modeling, 14(1), 2008. http://dx.doi.org/10.1142/S0218654308001051

[4] Edelsbrunner, H.; Harer, J.: Computational Topology: An Introduction, AMS, Providence, RI, 2010.

[5] Edelsbrunner, H.; Muecke, E.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Transactions on Graphics, 9(1), 1990, 66–104. http://dx.doi.org/10.1145/77635.77639

[6] Geiger, B., https://www-sop.inria.fr/prisme/logiciel/nuages.html.en, contour dataset repository.

[7] Geiger, B.: Three-dimensional modeling of human organs and its application to diagnosis and surgical planning, Ph.D Thesis, Ecole des Mines de Paris, France, 1993.

[8] Ju, T.; Warren, J.; Carson, J.; Eichele, G.; Thaller, C.; Chiu, W.; Bello, M.; Kakadiaris, I.: Building 3D surface networks from 2D curve networks with application to anatomical modeling. The Visual Computer, 21(8), 2005, 764–773. http://dx.doi.org/10.1007/s00371-005-0321-3

[9] O'Rourke, J.: Computational Geometry in C, second edition, Cambridge University Press, 1998. http://dx.doi.org/10.1017/CBO9780511804120