Title:
On the Nesting of a Contour Dataset

Authors:
John K. Johnstone, jkj@uab.edu, Computer and Information Sciences and Ophthalmology, UAB

Keywords:
Contour reconstruction, nesting, biomedical modeling.

DOI: 10.14733/cadconfP.2016.301-305

Nesting:

Contour datasets arise naturally in biomedical imaging and GIS, from CT/MR scans that model anatomy and topographic maps that model terrain (Fig. 1). The nesting of a contour inside another contour is an important property of a contour dataset. Nesting is important because it defines the inside of the shape. Contours at even nesting levels represent the object of interest, while contours at odd nesting levels represent holes. Since nesting captures the position of the inside of a shape relative to a contour, the nesting level of a contour dictates its treatment in many algorithms. For example, during reconstruction, contours at odd nesting levels should be connected only to other contours at odd nesting levels. 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 is to make the nesting analysis efficient and, most importantly, robust to dirty datasets. Contour datasets are often dirty, which complicates a nesting analysis (Fig. 2a). Fortunately, it turns out that a good nesting analysis can actually be used to repair the dataset while it determines nesting.

Most interesting contour datasets involve nesting. 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. 4688 of the 5012 contours in Barequet's mandible dataset *mnd_zm00* are nested. Dirty datasets are also common.

We now give rigorous definitions of the key concepts in nesting, the goal of our algorithm for nesting analysis, and the structure of the remaining sections of the paper.

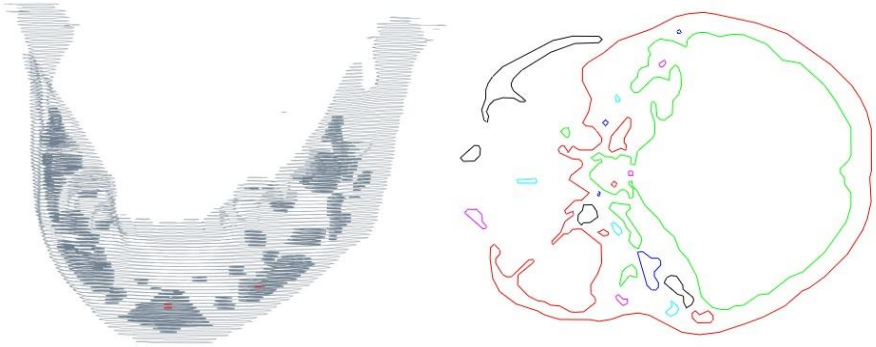

Fig. 1: (a) A contour dataset of a jaw and its nesting: contours of positive nesting level are highlighted. (b) Nested contours in a skull slice.

Definition 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 point sequences defining simple connected closed planar shapes called **contours**.

Since a contour dataset is sampled from sections of a simple orientable surface (an orientable 2-manifold without self-intersections), 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 individual contours are simple, the interior of a contour is well defined.) In practice, each of these relationships is often violated slightly, as we shall see, but we can view these as noisy versions of the Platonic ideal. This leads to the following formalization of nesting behaviour.

Definition 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.

Fig. 1b shows a slice with many contours of nesting level 1 and three contours of nesting level 2.

Definition 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. In principle, the boundary of a single contour does not self-intersect (i.e., contours are simple) and the interiors of two contours do not intersect, unless one contour lies completely inside the other. However, in practice, contour datasets often violate these assumptions.

Definition 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.

Note that the boundary of two contours may legally touch. Consider a critical section of the torus that generates a lemniscate. This lemniscate will be stored as two contours that touch, rather than a single self-intersecting contour. An example of dirty contours is given in Fig. 2a.

Dirty datasets are common, because of noise in image acquisition and segmentation. Note that the errors in a dirty dataset are usually limited: a contour will be close to simple, or two contours will overlap only slightly. However, the contour reconstruction literature uniformly assumes clean contour datasets (e.g., [2,3,7,8]). 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.

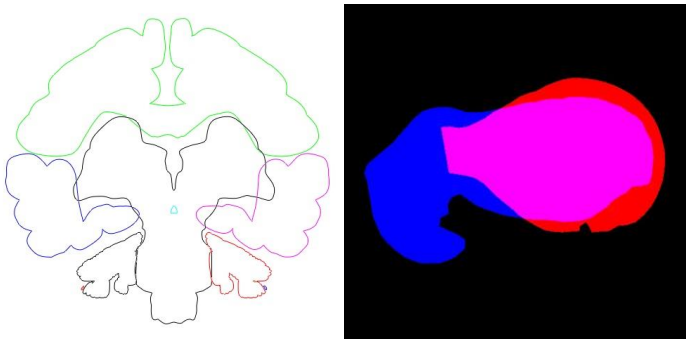

Fig. 2: (a) An example of dirty contours from a slice of a brain dataset. (b) Boolean operations in image space by filling. Intersection may be computed in image space by filling one polygon in red and the other polygon in blue: the intersection is now purple. The two differences are red and blue, while the union is red, blue or purple.

Nesting is an early analysis that must be done before any repair of a dirty dataset. To see how nesting analysis could be used for repair, note that, for example, a contour at nesting level *d* that overlaps its parent P at nesting level *d-1* should be pulled inside P during repair. Since correct repair of a dataset depends on nesting information, the contour dataset will be dirty during nesting analysis, in general. Therefore, nesting algorithms need to be resilient to this type of noise.

The rest of the abstract is structured as follows. After considering a naive algorithm to compute nesting level, and how it breaks down on dirty datasets, we give a robust algorithm to compute nesting level and nesting parent, followed by the image space algorithms for area that this nesting algorithm requires. A theme of the paper is the success of image space algorithms in handling dirty data.

Handling dirty datasets

If datasets were always clean, the computation of nesting level would be straightforward. (We ignore for now the more difficult computation of nesting parent.) 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 area should be incremented. Consider a possible algorithm using 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 of C and an arbitrary point d of D
if c lies inside D, increment the nesting level of C
else if d lies inside C, increment the nesting level of D

Unfortunately, this point location algorithm is brittle to the types of noise present in dirty datasets: if two polygons at the same nesting level slightly overlap and a point in the overlap is chosen as the candidate for point location, the two polygons 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 (see [5]). We see that the challenge is a robust implementation of nesting analysis in the context of dirty datasets.

Computing the nesting level and parent

A robust algorithm to compute nesting level uses area, as follows. In a preprocessing phase, compute the area a(C) of every contour C and the area of intersection a(C,D) of every pair of contours C, D in the same slice. An algorithm to compute the nesting level of every contour is as follows:

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? First note that there is a limit to the amount of noise that an algorithm can withstand. If two unnested contours mistakenly intersect 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. Therefore, this algorithm yields accurate results even in the presence of significant noise (overlap, violations of simplicity), since we are comparing to half the area of the smaller contour. The above algorithm for nesting level 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.

A crucial observation is that an image space algorithm for area of a polygon may be 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 efficient computation. Intersection in object space is difficult and sensitive to noise.

The other component of a nesting analysis is the computation of nesting parent. An algorithm to compute the nesting parent of every contour, which also uses area, is as follows:

```
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.

Image space algorithms for area

Consider image space algorithms to compute the area of a polygon and the area of intersection of two polygons, even if the polygons are dirty. Since only relative areas are needed for nesting analysis, it is fine that these areas are measured in pixels in image space. 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. 2b). 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.

Conclusions:

A robust and efficient algorithm to analyze the nesting of a contour dataset has been developed, even if the dataset is dirty. Most nontrivial datasets are nested and knowledge of nesting is important for their reconstruction. In the full paper, we will elaborate on the use of nesting in contour reconstruction and the use of nesting in the repair of a contour dataset.

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.

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 thankful to the community of scholars who have made contour datasets available, including the datasets of Bajaj, Barequet, Columbia, and Geiger that we use in this paper.

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.; Mucke, 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