Isocontouring with Sharp Corner Features

Sui Gong, Timothy S. Newman
Department of Computer Science, University of Alabama in Huntsville, Huntsville, USA
sg0010@uah.edu, tnewman@cs.uah.edu

Abstract. A method that achieves closed boundary finding in images (including slice images) with sub-pixel precision while enabling expression of sharp corners in that boundary is described. The method is a new extension to the well-known Marching Squares (MS) 2D isocontouring method that recovers sharp corner features that MS usually recovers as chamfered. The method has two major components: (1) detection of areas in the input image likely to contain sharp corner features, and (2) examination of image locations directly adjacent to the area with likely corners. Results of applying the new method, as well as its performance analysis, are also shown.

Key words: marching squares, feature preservation, corner recovery, contour finding, isocontours.

1. Introduction

In this paper, we describe an improvement for a popular means to find a closed boundary of a region of constant value (i.e., intensity or activity) in an image. This improvement can be considered to be an extension of the popular Marching Squares (MS) isocontouring method for 2D scalar fields (N.B. We have previously briefly described this new extension's key features in a conference report [9]). An isocontour can be defined as follows. Given a scalar field $f(x, y)$ (for example, f may represent the abstract function describing the densities captured in an X-ray), the isocontour is the collection of locations in the field having a particular scalar value α (e.g. the (x, y) locations where $f(x, y) = \alpha$). The scalar value α is the isovalue associated with (i.e., giving rise to) that isocontour. Isocontouring is a strategy often employed in processing or analyzing many types of data organized on grids. The most prevalent class of scalar 2D grid data is probably data arranged on rectilinear grids (such as X-ray images, slice images of CT datasets, intensity images, some planar simulation outputs, etc.). Other popular grid types include triangular and hybrid/adaptive grids [18, 29]. Here, our focus is on isocontouring for scalar data arranged on a rectilinear grid. Isocontouring is a very valuable technique for such data as it can, in one integrated process, define a closed boundary, with sub-pixel precision, of an area associated with some fixed level of activity (e.g. density).

There are a number of algorithms for finding isocontours in datasets of 2, 3 and more dimensions arranged on a grid. In determining the isocontour on these types of datasets, the existing algorithms treat the data values as samples from an underlying scalar field; such isocontouring algorithms form an isocontour in consideration of the samples. It is

often useful to find an approximate description of the boundaries of phenomena or structures in image processing and analysis applications. For example, boundary delineation can be useful for location-finding tasks, such as defect detection and front tracking. Also, isocontouring can be used to enable estimation of boundary or region properties (e.g. perimeter or area). In such uses, especially where a structure boundary is desired, the produced isocontour is sometimes said to be a reconstruction [17]. Other uses include as components of feature extraction [27] and segmentation [15] methods. Isocontouring can also be integrated into contour extraction frameworks [26]. Isocontours also have other uses, including discovery in terrain data [1, 12], simulation analysis [6] and fluid surface tracking [19].

The MS algorithm has been widely applied in 2D rectilinear grid data isocontouring. MS produces a piecewise linear approximation of the isocontour, as described in detail in Section 2. However, when applied in cases where the actual boundary has sharp corners, MS instead produces mostly blunted corners. This behavior of MS creates a problem for contour recovery when sharp corners are present. Producing a contour that recovers sharp corner features can improve renderings as well as assist in registration and pattern recognition. Thus, it is important for isocontouring to recover them correctly. However, previously there has not been a 2D isocontouring method that is able to correctly recover sharp corners. In this paper, we describe our extension to the Marching Squares that improves on that situation; our approach allows producing an isocontour that has sharp corner features. This extension enables better expression of actual boundary shape for boundaries containing such features.

This paper is organized as follows. Background is discussed in Section 2. Related work is described in Section 3. In Section 4, we present our extension to Marching Squares that enables construction of an isocontour that includes sharp corners. In Section 5, we provide results from applying the extended algorithm and comparisons with standard MS. The conclusion and future work are presented in Section 6.

2. Background

Marching Squares takes a 2D rectilinear grid of scalar data as its input. Such inputs could be X-ray images, individual slice images of volumetric datasets, infrared (IR) images, etc. The rectilinear grid is treated as a collection of grid cells by MS. In the case of images, each pixel value is treated as the data value at a grid point. The MS algorithm produces an isocontour with sub-cell accuracy by *marching* through the dataset in sequential order, processing one grid cell at a time until all grid cells have been processed. In each of the cells, the algorithm's processing involves a series of steps that determine if any isocontour segments need to be generated within the cell.

The first step MS follows in each cell is to compare each grid point's data value with the isovalue α. MS marks each data point with a value greater than or equal to α

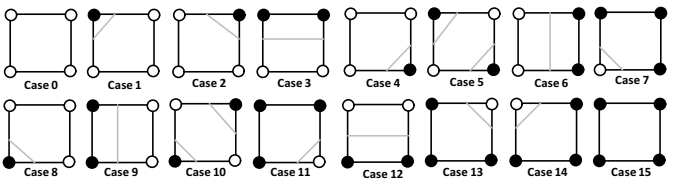


Fig. 1. 16 topological cases of cells in Marching Squares.

with a "1." It marks the others with a "0." Grid edges marked with a "1" at one end point and a "0" at the other are intersected by the isocontour.

There are $2^4 = 16$ possible markings for a cell since each grid cell has four grid points. Each unique marking type is called a topological case and defines a particular isocontour topology. The 16 cases and the general form of the contour segment(s) produced by MS for each case are shown in Fig. 1. In the figure, data points with a "1" marking are indicated by filled (black) grid points. Data points with a "0" marking are indicated by hollow (white) grid points. Typically, MS encodes the 16 possible cell cases in a look-up table before processing the dataset. For each case, that table records the identity of intersected edges and how the intersected edges are connected by the isocontour segments. When a look-up table is used, the second step of processing in the cell is to use the markings to determine the topological case for the cell.

Next, for each cell MS calculates positions of any isocontour intersections with the cell's grid edges. If a look-up table is used, the identities of intersected edges are retrieved directly from the table. For each such edge, linear interpolation on the values at the end points of the edge is used to find the intersection location.

Finally, in each cell, the isocontour is formed. In this step, MS connects each pair of intersection locations by a line segment. If a look-up table is used, the identities of the edges that are to be connected are found from it. The collection of such line segments defines the isocontour.

We note that two of the MS cases (i.e., Cases 5 and 10 in Fig. 1) can be contoured in a variant way from what is shown in Figure 1 (since there are two ways to connect each set of four intersections such that the contour does not self-intersect). Since triangular cells do not have this issue of variant contours (due to there being only one choice for each cell in a triangular mesh), in some cases isocontouring of rectilinear grid data is done by an analogue of MS that triangulates the cells before isocontouring (i.e., by dividing each cell into two triangles and then performing isocontouring in each triangle). This analogue can be called *Marching Diagonally Divided Squares* (MDDS).

One of MS's potential problems is that when sharp corner features occur in an actual boundary (e.g. of a structure), the isocontour that MS produces usually has chamfered corners at the places where the actual boundary has sharp corners, unless the location

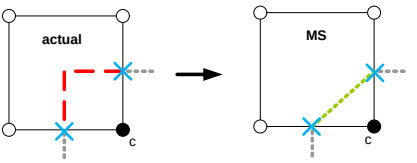


Fig. 2. A sharp corner of a boundary (**left**) that MS chamfers (**right**).

of the actual corner is very close to the edge of a grid cell. Fig. 2 shows an example of the problem. In the figure, a grid cell with three grid points whose values are less than the isovalue (i.e., denoted as hollow (white) circles in the figure) and one grid point whose value is greater than the isovalue (i.e., denoted as a filled (black) circle in the figure) is shown. The contour shown at the left (as a dashed line) represents the actual phenomenon boundary, which has a sharp corner feature. Such cells are recognized by the standard MS as Case 2 cells (using the numbering as in Fig. 1). The Case 2 topology specifies two grid edge intersections, and MS performs linear interpolation on these edges to find the intersection locations. For the Fig. 2 example, the $\times$ marks show the locations. MS connects the two intersections with a single line segment, as shown in the contour on the right (as a dotted line) of Fig. 2. The *corner* produced here by MS does not have the sharp shape of the actual boundary; instead, MS produces a chamfered corner.

The analogue of MS that triangulates each rectilinear grid cell before isocontouring in that triangulation can reduce the degree of chamfering for some corners. However, there are several costs for that benefit. First, the upfront triangulation of cells imposes an additional computational burden. Second, for an actual contour with a long, straight run that is diagonally-oriented, the contour produced for that *run* can be staircased. Third, the contour intersection locations on the diagonal edge segments of the triangulation differ from segment positions in MS since the analogue uses a different interpolation than the axis-aligned bilinear interpolant used by MS on the underlying rectilinear grid. Fourth, the total number of segments is usually significantly increased.

MS is often applied on many types of 2D grayscale images as a boundary-finding method (e.g. for segmentation [2, 29] or silhouette determination [11]). Compared to using contours or boundaries based on traditional, popular edge detectors, like the Sobel edge detector [10] (that typically produce edges that pass through pixel locations where the highest gradient changes are detected), using MS for boundary-finding has certain advantages. First, traditional edge detectors can produce a boundary that is not continuous (i.e., some individual edge pixels can be isolated (i.e., disconnected) from others), rather than a guaranteed continuous water-tight contour, as MS does. While edge detectors can be coupled with edge linking algorithms (e.g. such as [3, 17]) on the disconnected edge segments, the boundaries may still not be continuous. If the area needing the boundary description is known, one historically popular alternative approach to

determine a boundary description has been chain coding [7] (or its variants). However, in chain coding, the segment endpoints have pixel-level precision only. In comparison to traditional edge detection with linkage or chain coding, MS produces a contour that has sub-pixel-level precision. In addition, the MS contour segments have many possible orientations (limited only by the finite precision of machine floating-point arithmetic). In contrast, the elements of a chain-coded boundary can have only 8 possible orientations. Finally, the MS boundary is associated with a specific activity (or intensity) level whereas boundaries from edge linkages may not be associated with one activity (or intensity) level.

The boundary MS produces has been used in many applications, such as a part of processing to find the contour of a pelvic region in CT data [8], for generating toolpaths in a 3D printing application [28], and in a set relations visualization scheme [4]. Other than boundary finding, MS has been combined with methods for image segmentation [3] or with methods based on Minkowski functionals for image analysis [16]. In addition, MS has been used in a color distinguishability study to extract features such as stripe outlines from images [23]. MS can also be used in contour length estimation [3], since the contour segment produced for each cell is easy to calculate. A parallelization scheme for MS has also been described [20].

3. Related work in feature preserving isocontouring

Isocontouring that maintains certain shape features in the boundary it produces is called feature-preserving isocontouring. Next, some such existing approaches are described. These existing approaches are aimed at true 3D isocontouring (on volumetric data), usually by extension of the Marching Cubes (MC) Algorithm. MC can be viewed as a generalization of MS to volumetric data. Its isocontour is a surface called an *isosurface*. MC is very well-known [21, 22]. Like MS, MC can produce an isocontour that lacks certain "sharp" features, even if the isocontour represents the boundary of an underlying phenomenon or structure that does have such features.

One approach to enable certain sharp features to be preserved in an isosurface was described by Kobbelt et al. [14]. In their approach, after the isovalue is determined, the original dataset is converted into a directed distance field that stores three distance values at each grid point, namely the shortest distances in the x, y and z directions from the point to an isosurface component.

Then, Kobbelt et al.'s approach performs MC-like processing on the distance field as follows. First, a basic octree is built whose leaf nodes each store one 3D grid cell. The octree also records which cells are candidates for containing sharp features, as described later. The octree building starts from the leaf nodes. A merging process forms higher levels of the tree by merging some leaf nodes with their parent (if certain rules are satisfied). One such rule, for example, involves considering the current isocontour

position in lower level nodes versus the isocontour position in the next higher level node (i.e., formed by merging the lower level nodes). If the difference between these positions is too large, no merging takes place. Merging of non-leaf nodes with their parents occurs if they have particular values and all their children can also be merged. Following the merging process, nodes that haven't been merged are the candidate nodes (for containing sharp isocontour features). Grid cells within merged nodes are processed by standard Marching Cubes (since no sharp corner is likely present). In all cells of candidate nodes, the algorithm produces a different isocontour than MC; it creates additional triangular facets that could allow a sharp corner shape to be produced. These facets are connected to the ones generated in the nearby cells to avoid "holes" occurring in the isocontour.

Another feature preserving approach for 3D data is the fine feature recovery approach of Kaneko et al. [13]. By recovering fine features, they mean that the isocontour includes the thin or narrow parts of the actual underlying object or structure in a volumetric dataset. In the isocontour produced by standard MC, these types of features could be smaller or larger than they actually are. The Kaneko et al. approach first produces the standard MC contour. Then, using this contour, it estimates what the dataset values at each grid point location would need to be if the produced isocontour was the actual boundary. At each grid point, it then compares these estimates with the actual data value. Whenever such values differ significantly, the approach considers the recovered isocontour segments near this grid point to be in need of adjustment. It follows a two-step process to do that. First, it adds or subtracts a constant to the value at that grid point, producing an *adjusted dataset*. Then it produces an isocontour on the adjusted dataset using standard MC. The isocontour produced from the adjusted dataset encloses a region that has a volume that better resembles the volume enclosed by the actual boundary [13].

These two methods, however, are extensions of 3D isocontouring. They have not been extended to address sharp corner recovery in 2D isocontouring. In addition, while inspiring, the Kobbelt method has some overhead (i.e., it must produce the distance field prior to isocontour recovery), which can be time consuming. Kaneko's method, on the other hand, has a primary focus on fine feature preservation instead of sharp corner preservation. While sharp corners can sometimes also be fine features (e.g. when they are narrow and long), in most cases they are not fine enough for Kaneko's method to recover.

Our method described here extends the state-of-the-art for 2D isocontouring by allowing sharp corner recovery in 2D isocontouring.

4. Methods: Corner Feature Expressive Marching Squares

Our new algorithm is a feature-preserving approach that extends the Marching Squares to allow expression of sharp corner features. In this section, a step-by-step elaboration of our algorithm and an enumeration of its topological cases are described.

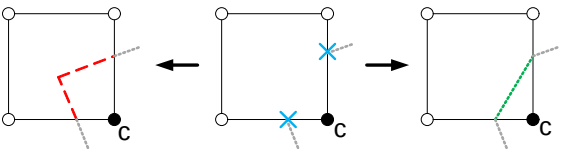


Fig. 3. Two possible contours. (**Left**) contour with a sharp corner feature; (**right**) contour with a chamfered corner feature; (**middle**) with the same intersection locations.

4.1. Corner Feature Expressive Marching Squares: motivation

Our corner feature expressive algorithm can produce a contour with sharp corners using information from small, local regions surrounding the corner. We motivate the algorithm's approach next. Later in the section, the specific, step-by-step processing is laid out.

As discussed in Section 2, the standard MS has, for each topological case, a single predefined way to connect the points of intersections of the isocontour with the grid lines. Thus, no matter what type of contour shape is actually present in a given cell, the predefined contour shape for that cell's topological case is the only type of shape that can be produced for the isocontour for the cell. An example of this situation is shown for one cell (of topological Case 4) in Fig. 3. For the case of this figure, the isocontour has a segment below the cell, which starts from the middle of the lower edge and extends down and to the right, and a segment on the right of the cell, which starts in the middle of the right edge and extends above and to the right (i.e., these parts of the isocontour are shown as dotted lines in the center part of the figure). The lower right lattice point of this cell is labeled as C. The value at that lattice point is used in determining both of the isocontour's intersection locations with the cell. Here, those points are marked with "$\times$" marks in the center part of the figure. The contour MS produces is shown as a dotted line on the right. This contour has a chamfered corner, even though the true contour shape could include a sharp corner, like the one shown on the left of this figure. Using only the locations of the isocontour intersections with the grid lines bounding the cell, it is not possible to tell whether the shape of the contours inside the grid cell contains a chamfered corner shape or a square corner shape.

Our algorithm exploits the contour segment orientations in the cells adjacent to possible corners to estimate contour shape and corner position in cells possibly containing a corner. In particular, it first considers if the contour segments in the adjacent cells provide clues suggesting if there could be a sharp corner.

An example of two sorts of clues that neighboring cells can provide is shown in Fig. 4. In the middle part of this Figure, two types of contours with different corner shapes are shown for one cell. Beside that, two possible arrangements of cells neighboring the one in

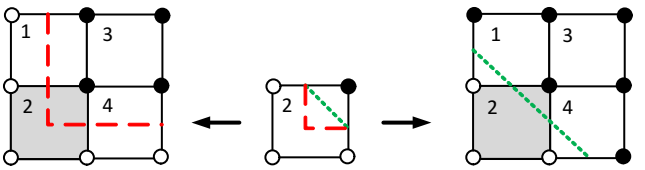


Fig. 4. Two possible arrangements of cells neighboring one grid cell.

question are shown. The cell in question is shaded and numbered "2." The neighboring cells in this example are labelled "1," "3" and "4." The actual contour shapes are also shown for each arrangement. In both arrangements shown here, the cell in question has the Case 2 topology. One arrangement features a diagonal edge (perhaps part of a chamfered corner), as shown by the dotted line. The other one features a sharp corner, as shown by the dashed segments. Since the grid cell in question has a Case 2 topology, the isocontour in it will be produced as a diagonal edge in standard MS, regardless if there is in fact a chamfered corner there. However, here the neighboring cells can provide some clue about the likely corner type in the cell. For the arrangement shown at the left, the dashed contour in the cells labeled "1" and "4" provides a clue that there may be a sharp corner feature in the shaded cell. For that arrangement, cells "4" and "1" have the Case 3 and Case 6 topologies, respectively. For such scenarios, a contour similar to the square corner shown in the left arrangement seems more credible to many human observers. In contrast, for the arrangement shown at the right, the dotted contour in the cells labeled "1" and "4" does not provide such a clue. For that arrangement, cells "1" and "4" have Case 7 topologies, and the contour produced by MS has a suitable shape for the cell in question.

4.2. Corner Feature Expressive Marching Squares: elaboration

In keeping with such reasoning, our extended Marching Squares considers cells adjacent to potential corner-containing grid cells to determine situations for which sharp corners are credible. Our approach follows processing similar to the illustration shown for the situation in Fig. 4; cells likely to contain a part of the boundary that has a sharp corner are found by considering the neighborhood about the cell. We call such possible corner-containing cells *candidate corner cells*, and we call neighborhoods about the corner candidates *grid cell groups*. Each grid cell group consists of the candidate corner cell and two cells adjacent (i.e., that are 4-neighbors) to that one. Specifically, the two adjacent cells considered are the ones that the contour enters from the possible corner-containing cell. (Due to the way MS finds isocontour-cell intersections, the adjacent cells can only be horizontally or vertically adjacent to the candidate corner cell.) We call those two adjacent cells the adjacent neighbor cells.

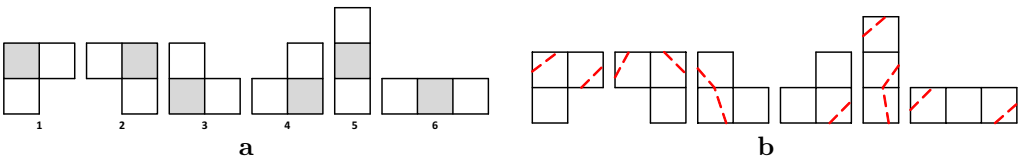


Fig. 5. Grid cell group layouts and insufficient groups. (**a**) Enumeration of grid cell group. (**b**) Insufficient (grid cell) groups.

There are 6 possible layouts (forms) for each grid cell group, as shown in Fig. 5a. We call each one a grid cell group layout, and we label them as Layouts 1 through 6. In Fig. 5a, the candidate corner cell for each layout is shaded. The other two grid cells that are not shaded are its adjacent neighbor cells. Each grid cell group has a total of eight grid points and ten unique edges. Since each grid point can either be marked as "1" or "0", there are $2^8 = 256$ possible different combinations of markings. However, in corner detection processing, it is not necessary to consider such a number of combinations since some of the combinations do not contain a corner (e.g. the combinations with all 0 or all 1 markings have no cells intersected by the isocontour).

Grid cell groups that include diagonally-adjacent cells could potentially be used, but we do not consider them here, since by themselves they often do not provide additional credible evidence of corners. But, if a larger set of nearby cells was considered, these larger groups of grid cells could be useful for corner detection. We leave corner detection and sharp corner contour production for such groups to future work; we focus here on a methodology that is applicable to grid cell groups of size 3. In the work here, evidence from size 3 grid cell groups is used to detect where sharp corner features are likely and then to produce contours containing such features. We also focus only on grid cell groups that have sufficient neighbor information.

Fig. 5b shows six example grid cell groups with insufficient neighbor information. In it, each group contains at least one part of a contour that might in fact be a sharp corner but the candidate corner cells in the group do not have two adjacent neighbors that both contain continuations of the contour segment from the candidate corner cell. We term such grid cell groups without enough neighboring information as *insufficient groups*. We call grid cell groups that have enough neighboring information *sufficient groups*.

Some sharp corners in insufficient groups are in fact detected and produced correctly by our method. This detection occurs due to a nearby sufficient group that includes some grid cells that are also in the insufficient group. An example of such a situation is shown in Fig. 6b. In this figure, there are three grid cell groups that are highlighted (using an outline feature or by gray shading). One of them (the gray shaded one) can be considered as an insufficient group. The other two (shown as the solid outline and dashed outline) are sufficient groups. The insufficient grid cell group contains two corner

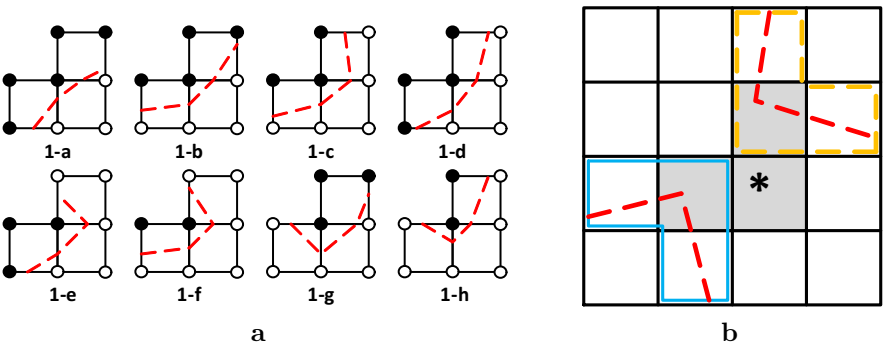


Fig. 6. Examples of group topology and insufficient/sufficient groups. (**a**) Group topology list for Case 1 candidate corner cell. (**b**) Corner features detectable from the sufficient groups (dashed outlined and solid outlined) but not in the insufficient group (shaded).

features, but neither occurs at its candidate corner cell (marked by "$*$"). However, each corner feature can be detected via one of the two (outlined) sufficient grid cell groups since they offer credible evidence that a corner could be present.

We examined all of the 16 topological cases for MS and determined the grid cell group topologies that could contain sharp corners. Our algorithm stores such cases in a group topology list. In this list, each group contains the three grid cells making up a grid cell group. These are the candidate corner cell and its opposing neighbor cells. Each MS topological case, except Cases 0 and 15, can give rise to a candidate corner cell. Thus, for the Case 1 through 14 topologies, we examined all of the 4-neighbors of candidate corner cells and the contour segments in them.

We list some examples of the group topologies in Fig. 6a. In this figure, each group topology is labelled with a combination of a number and a letter. The number indicates the case topology of its candidate corner cell in the original MS look-up table (from Fig. 1), and the letter indicates our subcase identifier. For example, in Fig. 6a, we list all 8 subcases for candidate corner cells with Case 1, labeled as 1-a through 1-h. The complete group topology listing for all the cases is shown in Appendix A, and that listing is labelled with the same scheme as in Fig. 6a.

While our method can produce a contour in the candidate corner cell that differs from what standard MS produces for that, it does not vary the contour shape in the other cells of the grid cell group. To detect if the contour in a candidate corner cell is likely to contain a sharp corner, we first determine the orientations of the contour segments in the adjacent neighbor cells of its grid cell group. Then the angular difference between these orientations is calculated. If that difference is close to 90 degrees, our method marks the

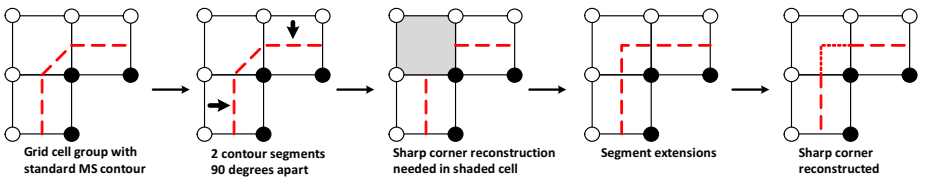


Fig. 7. Corner detection and production process.

cell as one which likely has a sharp corner feature. Otherwise, when the angle between two line segments in the neighbors is not sharp, we classify the cell as a non-corner cell and use the standard MS rules to produce the isocontour segments. In addition, all other non-corner cells are produced according to the standard MS rules.

For candidate corner cells that likely have a sharp corner, a two-step process is used to produce the corner. First the corner location is determined as the point where the contour segments in the adjacent neighbor cells intersect. Then, contour segments are formed connecting this point to the locations where the contours intersect cell edges.

Fig. 7 shows an example of the corner construction process for a grid cell group with a Case 4 candidate corner cell. In the adjacent neighbor cells, the two contour segments form an angle of 90 degrees. In this case, due to the size of the angular difference, our method determines that the candidate corner cell contains a sharp corner feature (i.e., thus, for that cell, we do *not* use standard MS rules to produce its contour). Once a likely sharp corner has been determined, contour segments in opposing neighbor cells are extended, as shown at the right part of the figure. The final result is the contour with the sharp corner shown in the last panel of the figure.

4.3. Corner Feature Expressive Marching Squares: algorithm

A step-by-step elaboration of our algorithm is shown in the listing labelled Algorithm 4.1.

5. Results and discussion

Next, some results of applying our algorithm in images are presented. Results of synthetic images are presented first, followed by images from volumetric datasets, and then X-ray images at last. For comparison, results for standard MS are also exhibited.

Our synthetic images are density/occupancy images of objects with density 100 in a 16×16 rectilinear grid. The background density is 0. At each grid lattice point, the data value is the aggregate density over a unit-sized square region centered at that lattice point. Thus, any lattice point representing a fully occupied unit-sized region in space

Algorithm 4.1 Corner Feature Expressive Marching Squares.

```
for each grid cell in the dataset do
    determine its topological case
    if it is not Case 0 or Case 15 then
        locate its two adjacent neighbor cells according to the group topology list
        calculate the contour segment orientations in the neighboring cells
        if they form a sharp angle then
            extend the contour segments of the adjacent neighbor cells into this cell to
produce a sharp corner
        else
            apply standard MS on the cell
        end if
    else
        apply standard MS on the cell
    end if
end for
```

has a data value of 100 associated with it. The objects are axis-aligned in some of the images, but not in others. We can determine the error in an isocontouring method's result using these images because the actual boundaries in each of them are known.

A comparison of results from standard MS and our algorithm for synthetic images of an L-shaped block, star-shaped object (in two orientations), and a ninja star-shaped object is shown in Fig. 8. In this figure, the dashed contours are the results produced by our algorithm and the dotted contours are the results produced by standard MS. The solid black contours are the actual object boundaries. The parts of the boundary that lack sharp corner features yield overlapping contours from standard MS and our algorithm.

We also show a result from standard MS, our algorithm, and MDDS in Fig. 9b. In this figure – and for Fig. 10, 11 and 12 – we use the same contour drawing patterns as used in Fig. 8, except we periodically overlay circle glyphs on the dashed results (of our approach). In addition, the result of MDDS is drawn in a dash-dot pattern in the Fig. 9b. Our approach recovered four sharp corners while MDDS recovered none. MDDS also recovered some of the straight edges as wavy edges. Since MDDS does not produce a competitive result for sharp corner recovery, it is not shown in other result images in this paper.

Our new algorithm recovered many, but not all, of the sharp corners in these images. In particular, if a corner feature spans a few cells, the new algorithm can have difficulty to fully recover all corners. Expanding the size of the grid cell groups may improve matters, although possibly at a cost of false positives.

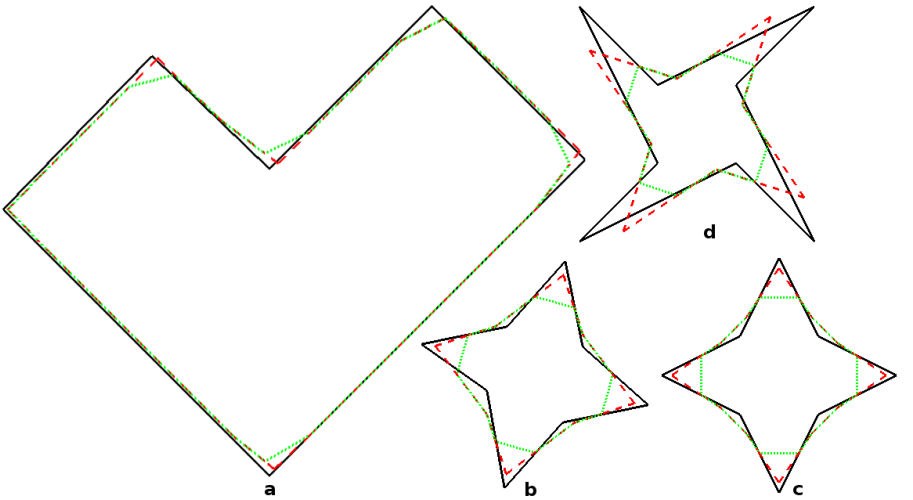


Fig. 8. Synthetic dataset of object isocontouring results for two approaches. (**a**) L-shape oriented at 45 degree angle with the x-axis. (**b**) A star object oriented at 15 degree angle with the x-axis. (**c**) A Star object oriented at 0 degree angle with the x-axis. (**d**) A ninja star object.

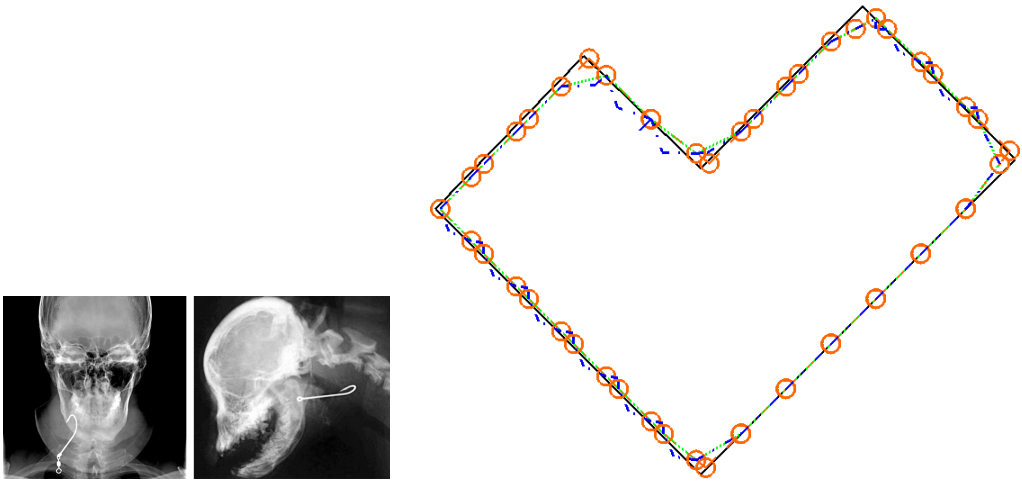


Fig. 9. Some applications. (**a**) X-ray images of: (left) hook embedded in a human face, (right) hook caught on a dog. (**b**) Example results.

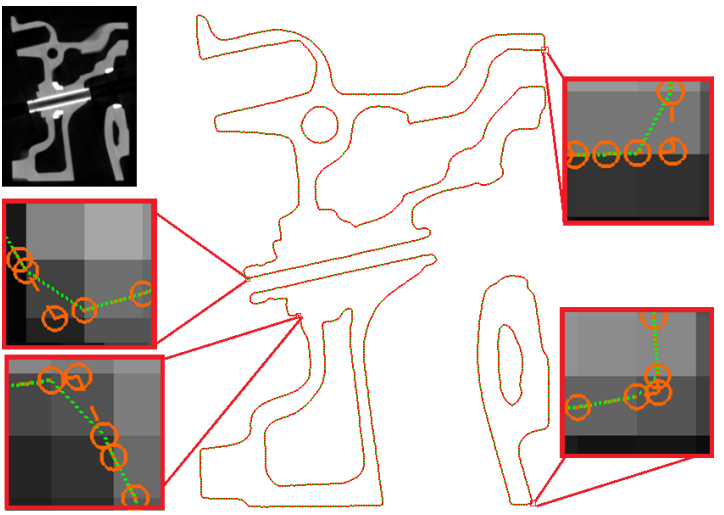


Fig. 10. Engine Block dataset slice image isocontouring results. (**Upper left corner**) the slice image.

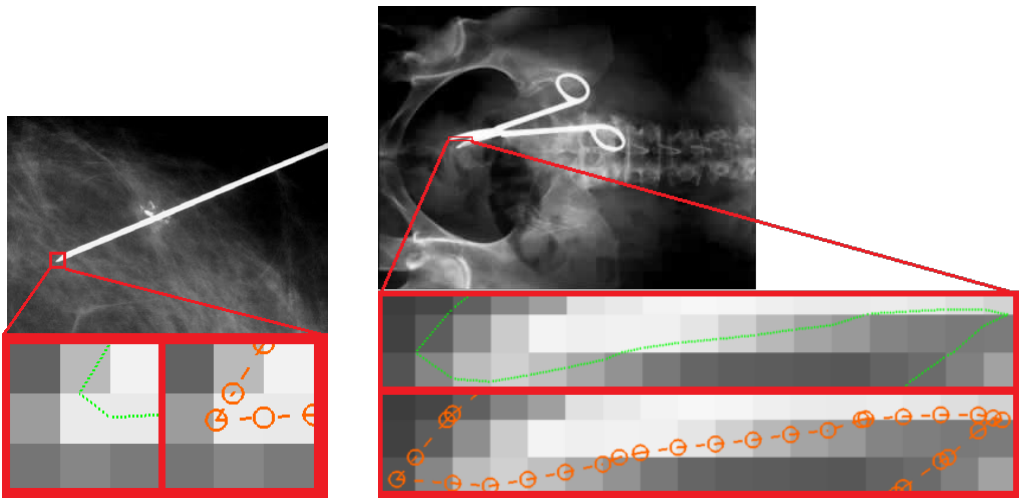
Next, we describe results from applying our algorithm on real images. First, results for one slice image (size 256×256) of the well-known Engine Block dataset (from the Volume Library [25]) are shown in Fig. 10 using the line marking pattern described earlier. In this figure, zoomed-in call-outs are shown for several corners. The gray scale image of this slice of data is also shown at the upper left of the figure as a reference.

We applied our algorithm and standard MS on 10 of the slices of the Engine Block dataset and counted the number of sharp corners recovered by each algorithm. The corner counts are presented in Tab. 1. On average, our algorithm recovered 10 sharp corners more than standard MS did.

Results for applications to some X-ray images are shown in Fig. 11, which includes comparison results for standard MS versus our algorithm for two images containing objects with sharp corners (from radiopaedia.org [5]). The Fig. 11a image is of a biopsy needle is size 220×320. The Fig. 11b image is of scissors lodged in a human and is size 320×240. We call these images Needle and Scissors, respectively. For each image,

Tab. 1. Number of sharp corners recovered by our algorithm but not by standard MS.

Image No.	1	2	3	4	5	6	7	8	9	10
Sharp Corners	13	12	10	15	10	9	14	7	4	6



a **b**

Fig. 11. X-ray image and isocontour comparison results: (**a**) biopsy needle, (**b**) scissors lodged inside human.

a zoomed-in call-out of an area containing a sharp corner feature is shown, with the result of standard MS (in dotted segments) side-by-side with our algorithm result (in dashed segments). In these images, the new algorithm has produced a sharper *point* for the needle and scissors objects than standard MS has. We also applied both algorithms on four images (labelled Hook1, Hook2, Hook3, and Hook4) of hooks embedded in tissue, two of which are shown in Fig. 9a.

Some results for applying our algorithm to three range images of block or box objects containing sharp corners are shown in Fig. 12. These images are from the OSU (MSU/WSU) range image database [24] and include a set of rectangle blocks (called Blox2), a set of cylindrical blocks (called Blox3), and three stacked boxes (called Stack). The Blox2 and Blox3 images are size 240×240. The Stack image is size 128×128. For each image, we exhibit several zoomed-in call-outs of areas containing sharp corner features, showing both the result of standard MS (in dotted segments) and our algorithm (in dashed segments) shown. In these images, the new algorithm has produced many sharp corners while MS has failed to produce them.

5.1. Empirical Error Studies: Our Algorithm vs. Standard MS

We have also used synthetic images in empirical tests of our algorithm and standard Marching Squares to study contour accuracy in a quantifiable way. In these images, there

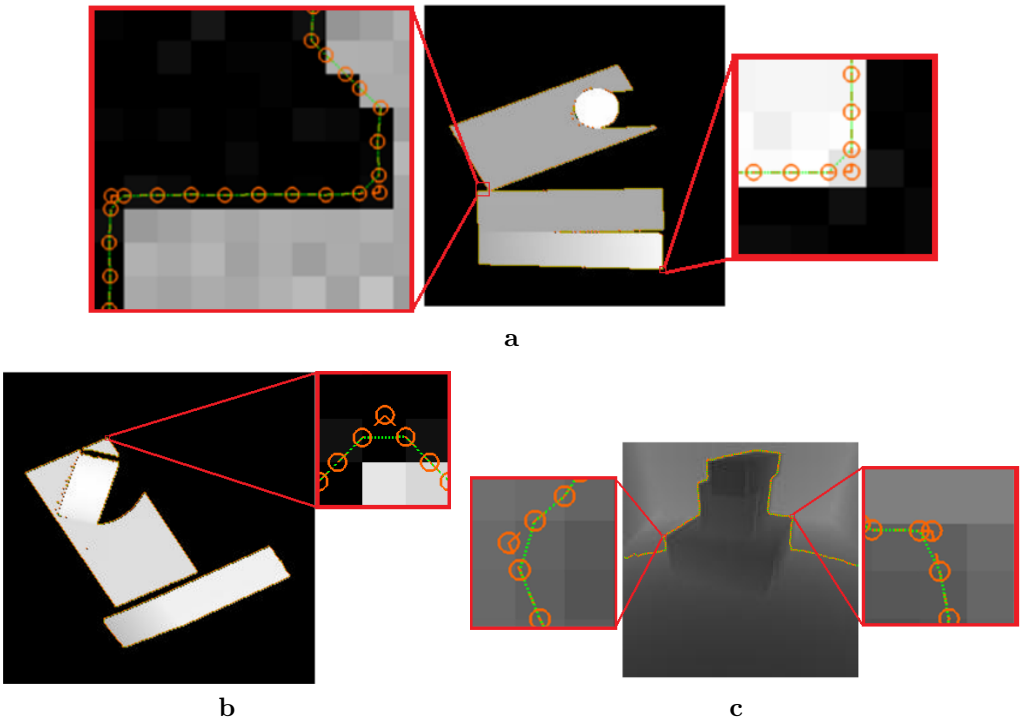


Fig. 12. Range images and isocontour comparison results. (**a**) Blox2. (**b**) Blox3. (**c**) Stack.

are sharp corners that have an internal angle very close to 90 degrees. In determining accuracy for the tests, the lower left grid point of each cell was taken as a local coordinate system origin of a cell assumed to be size 1×1. A number of test scenarios were used in the experiments, with the sharp corner locations varied in each scenario. Four methods were used to estimate the error in the contours. Those methods and outcomes from using them are presented next.

5.1.1. Average Closest Euclidean Distance Error Measure

For each test case, we determined the smallest Euclidean distance to the actual boundary from points sampled from the recovered contour. The average of these distances was taken as the error E_D in the recovered contour for that test case:

$$E_D = \frac{1}{N} \sum D(i)\,, \tag{1}$$

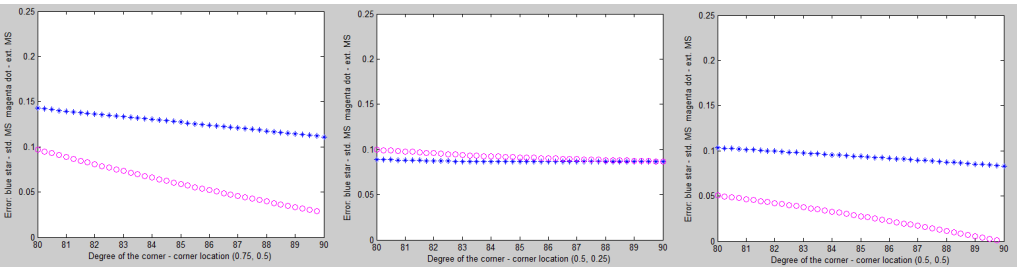


Fig. 13. Average closest Euclidean distance error measure, E_D, for new algorithm (circles) and standard MS (stars).

where N is the number of sample points, and $D(i)$ is the closest distance from the i-th sample point on the recovered contour to the actual boundary.

Plots of such errors for a range of test cases are shown in Fig. 13. These plots consider test cases for three distinct corner locations (the center of the cell and two other points on the diagonal of a cell, as indicated in the plot captions). For each plot, the x axis indicates the degree of the corner, and the y axis indicates the error for each case. The stars show error for standard MS, and the circles show error for the new algorithm. For most cells with sharp corners, the new algorithm yielded the lower error. Its improvement appears to be best when corner angles are close to 90 degrees.

5.1.2. Corresponding Points Error Measure

Another error measurement we considered was E_{DC}, an average distance measure computed by averaging distances between corresponding points on two contours. Since standard MS does not usually produce a sharp corner feature, we determined the distance measure's component for the corner by taking the mid point of MS's contour as the point corresponding to the corner of the actual contour. We then divided it into two pieces at its mid point, with each piece corresponding to one portion of the contour that forms the sharp corner. Then, evenly-spaced point samples were taken on each part of the contour. The same number of point samples were also taken on the actual corner boundary, and they were also evenly-spaced. The correspondences of the two sets of points were then found one-for-one (e.g. the first point on the recovered boundary corresponded to the first point on the actual boundary, etc.)

We tested nine cases of distinct corner locations inside a cell using this error measure. Results are shown in plots in Fig. 14. In each test case, the location of the corner was fixed as indicated in each plot (e.g. in the first plot, the corner location is $(0.5, 0.5)$ – the center of the cell). For each location, the degree of the corner varied from 80

to 90 degrees. The stars show error for standard MS and the circles show error for the extended method. For the cases shown in Fig. 14, the isocontour produced by our algorithm had a lower error than the isocontour produced by standard MS.

5.1.3. Contour Length Ratio Error Measure

A third measure we considered was the length of the produced contours versus the length of the actual boundary. We call this measure the *contour length ratio error measure*. The measure uses the ratio of the length of the recovered isocontour, L_r, to the length of the actual boundary, L_a. The error of this measure, E_L, is defined as:

$$E_L = L_r / L_a \,. \tag{2}$$

Thus, values closer to 1 represent better contours. We used the same test cases described at the beginning of Section 5.1 to find this error measure, and we show several plots of error in Fig. 15, with the circles denoting the error of our method and the stars denoting the error of standard MS. The average value of E_L for standard MS is 0.55 (45% underestimation) while the average value of E_L for our algorithm is 0.77 (23% underestimation). These experiments suggest that the extended method produces a contour that has a length closer to the actual boundary than does standard MS.

5.1.4. Error between Actual Corner Location and Recovered Corner Location

A fourth error measure we considered is the discrepancy (error) between the actual corner location and the recovered corner location. We consider such error versus the corner's relative distance to a reference location, L_{AC}, on the grid cell. Since standard MS does not recover a sharp corner, we considered the midpoint of the contour as the recovered corner location, L_{RC}, to measure the discrepancy. The error of this measure, E_{CL}, is defined as:

$$E_{CL} = |L_{AC} - L_{RC}| \,. \tag{3}$$

To evaluate this error, we used synthetic images of an object with a 90-degree sharp corner inside one grid cell. We varied the corner location in each test image.

To determine E_{CL}, we used a grid cell *vertex* as the reference location. Specifically, we used the vertex that is closest to the recovered corner. A plot of error measured this way is shown in Fig. 16a. In this figure, the errors of our algorithm are shown as circles, and the errors of standard MS are shown as stars. Our algorithm was found to produce an error free contour when the actual location is at the center of the grid cell (at that location, the distance to the reference location is 0.71).

We also computed E_{CL} using a location of the grid cell *edge* closest to the recovered corner as the reference location. A plot of error measured this way is shown in Fig. 16b. This plot uses the same color coding as Fig. 16a does. The error free corner case can also be observed in this figure. These two experiments suggest that our algorithm produces

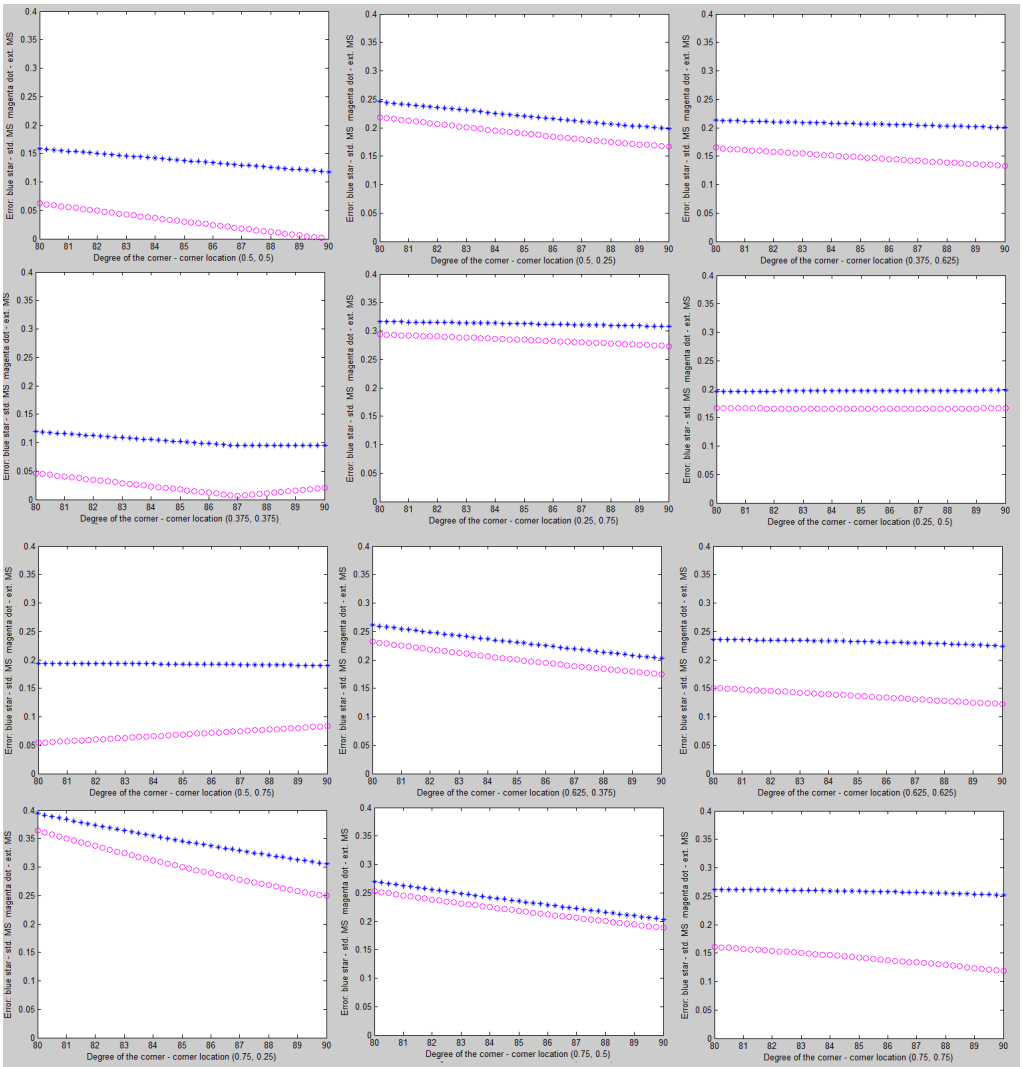


Fig. 14. Average distance between corresponding points error measure, E_{DC}, for new algorithm (circles) and standard MS (stars).

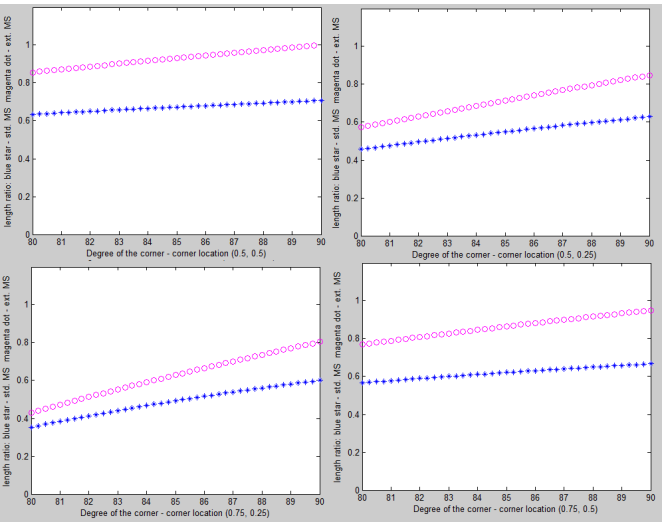


Fig. 15. Contour length ratio error, E_L, measure for new algorithm (circles) and standard MS (stars).

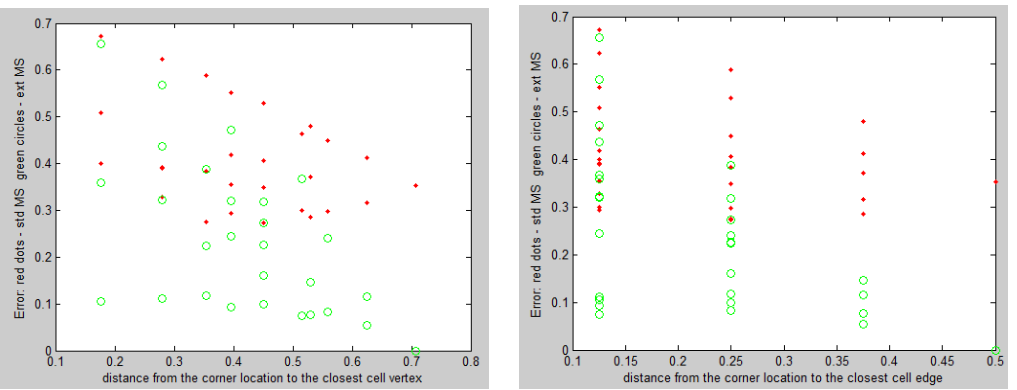


Fig. 16. The distance between recovered corner and the actual corner, E_{CL}, vs. the distance between the actual corner location and two different reference locations. (**a**) Reference location: the closest grid cell *vertex*. (**b**) Reference location: the closest grid cell *edge*.

a corner that is located closer to, and sometimes at the exact location of, the actual corner, while standard MS invariably exhibited error. Our algorithm also tends to produce corners with error smaller than the contours produced by standard MS.

5.2. Processing Time

Next, we report on processing times for the new algorithm and the standard MS. Timings were done on a machine with an Intel i7-3770 quad core processor and 12GB of memory. The datasets used for testing were the Hook1, Hook2, Needle, and Scissors images described earlier and the synthetic L-shaped axis aligned block used in Fig. 8 (which we call Synthetic here). Results are presented in Tab. 2. In these tests, our method had slightly longer processing times than the standard MS due to its additional processing steps for the corner-containing grid cells, but the amount of overhead associated with sharp corner production is quite small (averaging 3.3% overhead).

6. Conclusions

We have presented a new algorithm that allows for production of an isocontour with sharp corner features. The method is an extension to the Marching Squares algorithm. It exploits contour information from neighboring cells to determine likely locations of sharp corners and then creates a contour with sharp corners there. Applications of the new algorithm on synthetic datasets and X-ray images suggest that when sharp corner features occur in the actual boundaries, the new algorithm produces contours closer to the actual boundaries than standard MS does. Error studies show that our approach exhibits smaller error than standard MS in (1) two out of three cases using average closest Euclidean distance, (2) all twelve cases using average distance between corresponding points, and (3) all four cases using the contour length ratio measure. The new algorithm also produces very small to no error when the angle of the sharp corner is close to 90 degrees.

In future work, we hope to develop further extensions that can produce other contour feature types.

Tab. 2. Processing times.

	Processing times [ms]				Processing times (in ms)		
Dataset	Std. MS	Our alg.	Overhead	Dataset	Std. MS	Our alg.	Overhead
Synthetic	0.132	0.143	7.7%	Hook4	0.138	0.144	4.2%
Hook1	0.142	0.147	3.4%	Needle	0.131	0.135	3.0%
Hook2	0.143	0.145	1.4%	Scissors	0.138	0.141	2.1%
Hook3	0.142	0.144	1.4%				

A. Grid Cell Group Topologies

Figs. 17 and 18 show the grid cell group topologies keyed to the Cases 1 to 14 for Marching Squares.

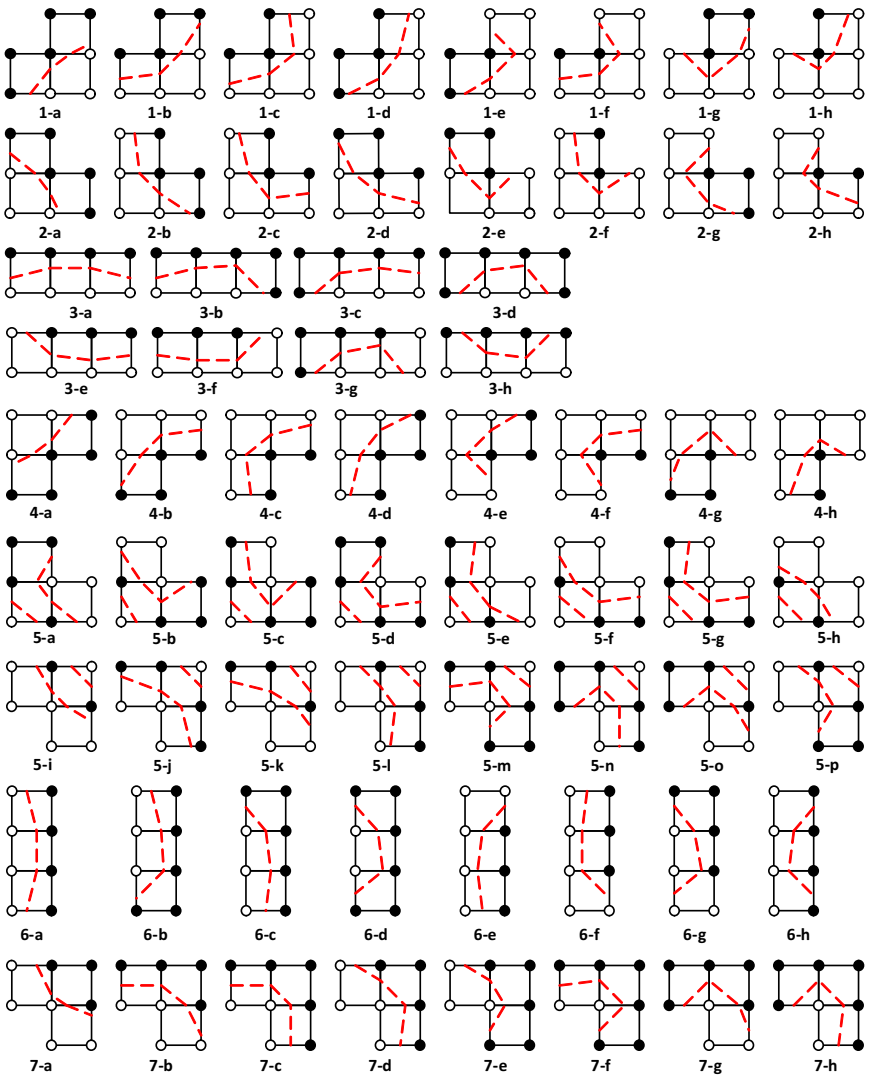


Fig. 17. Grid cell group look-up table – part I.

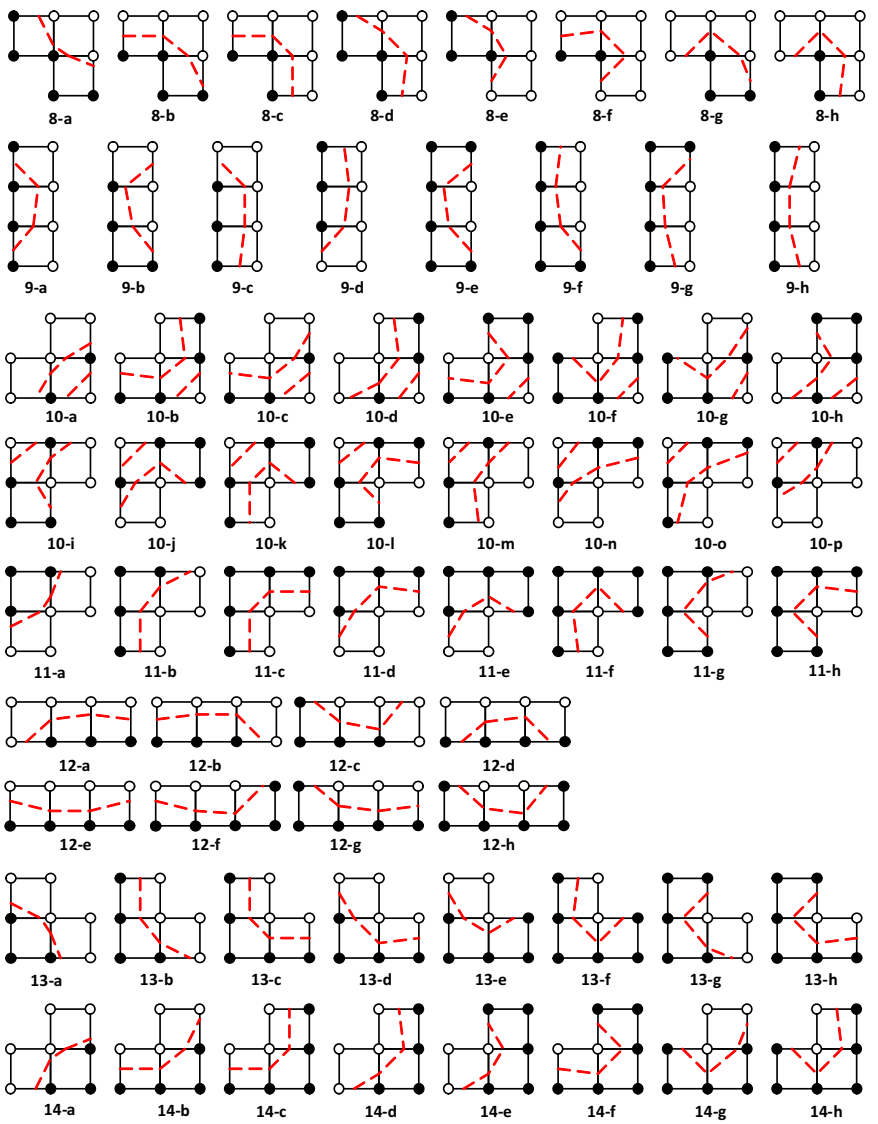


Fig. 18. Grid cell group look-up table – part II.

References

[1] M. Cammarano. Depicting terrain with shaded relief maps. Stanford Univ. Class Report, 2004. http://graphics.stanford.edu/~mcammara/vis2004/paper.pdf. Accessed: Sept 22, 2015.

[2] P. B. Chamberlain and C. L. Thomas. Direct thick layer rapid prototyping from medical images. In *Proc. 10th Solid Freeform Fabrication Symp. SFF '99*, pages 599–605, Austin, TX, August 9-11, 1999. http://sffsymposium.engr.utexas.edu/Manuscripts/1999/1999-069-Chamberlain.pdf.

[3] M. P. Cipolletti, C. A. Delrieux, G Perillo, and M. C. Piccolo. Superresolution border segmentation and measurement in remote sensing images. *Computers & Geosciences*, 40:87–96, March 2012. doi:10.1016/j.cageo.2011.07.015.

[4] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. and Comp. Graphics*, 15(6):1009–1016, November 2009. doi:10.1109/TVCG.2009.122.

[5] Fishing accident. http://radiopaedia.org/cases/fishing-accident. Accessed: September 22, 2015.

[6] A. Fofonov, V. Molchanov, and L. Linsen. Visual analysis of multi-run spatio-temporal simulations using isocontour similarity for projected views. *to appear in IEEE Trans. Vis. and Comp. Graphics*, pages 2037–2050, 2016. doi:10.1109/TVCG.2015.2498554.

[7] H. Freeman. Computer processing of line-drawing images. *ACM Comput. Surv.*, 6(1):57–97, March 1974. doi:10.1145/356625.356627.

[8] Q. Gao, S. M. Ali, and P. Edwards. Automated atlas-based pelvimetry using hybrid registration. In *Proc. IEEE 10th Int. Symp. Biomedical Imaging ISBI 2013*, pages 1292–1295, San Francisco, USA, April 2013. doi:10.1109/ISBI.2013.6556768.

[9] S. Gong and T. S. Newman. A corner feature sensitive marching squares. In *Proc. IEEE Southeastcon SECON 2013*, pages 1–6, Jacksonville, USA, April 2013. doi:10.1109/SECON.2013.6567363.

[10] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 3 edition, 2007.

[11] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1134–1141, Los Angeles, CA, USA, 2005. ACM. doi:10.1145/1186822.1073323.

[12] L. S. Johnson. Progressive transmission of surfaces with geometric constraints. Master's thesis, Univ. of South Carolina, 2004.

[13] T. Kaneko and Y. Yamamoto. Volume-preserving surface reconstruction from volume data. In *Proc. Int. Conf. Image Processing ICIP '97*, volume 1, pages 145–148, Santa Barbara, USA, October 1997. doi:10.1109/ICIP.1997.647405.

[14] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H-P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 57–66, New York, NY, USA, 2001. ACM. doi:10.1145/383259.383265.

[15] K. R. Krishnan and S. Radhakrishnan. Focal and diffused liver disease classification from ultrasound images based on isocontour segmentation. *IET Image Proc.*, 9(4):261–270, 2015. doi:10.1049/iet-ipr.2014.0202.

[16] H. Mantz, K. Jacobs, and K. Mecke. Utilizing Minkowski functionals for image analysis: A marching square algorithm. *J. Statistical Mechanics: Theory and Experiment*, 2008(12):P12105, 2008. doi:10.1088/1742-5468/2008/12/P12015.

[17] C. Maple. Geometric design and space planning using the marching squares and marching cube algorithms. In *Proc. Int. Conf. Geometric Modeling and Graphics GMAG 2003*, pages 90–95, July 2003. doi:10.1109/GMAG.2003.1219671.

[18] P. Moinier, J-D. Müller, and M. B. Giles. Edge-based multigrid and preconditioning for hybrid grids. *AIAA Journal*, 40(10):1954–1960, 2002. doi:10.2514/2.1556.

[19] M. Müller. Fast and robust tracking of fluid surfaces. In *Proc. 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 237–245, New Orleans, Louisiana, USA, 2009. ACM. doi:10.1145/1599470.1599501.

[20] A. Murthy, E. Bartocci, F. Fento, et al. Curvature analysis of cardiac excitation wavefronts. *IEEE Trans. Computational Biology and Bioinformatics*, 10(2):323–336, 2013. doi:10.1109/TCBB.2012.125.

[21] T. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers and Graphics*, 30(5):854–879, 2006. doi:10.1016/j.cag.2006.07.021.

[22] G. M. Nielson. On marching cubes. *IEEE Trans. Vis. and Comp. Graphics*, 9(3):283–297, July 2003. doi:10.1109/TVCG.2003.1207437.

[23] Y. Omori, T. Murakami, and T. Ikeda. Color universal design without restricting colors and their combinations using lightness contrast dithering. In *Proc. 5th Int. Congress of Int. Assoc. of Societies of Design Res. IASDR 2013*, 2013. Paper No. 2227-1. http://design-cu.jp/iasdr2013/papers/2227-1b.pdf.

[24] OSU (MSU/WSU) range image database. http://web.archive.org/web/19991008150305/http://eewww.eng.ohio-state.edu/~flynn/3DDB/RID/. Accessed: October 23, 2016.

[25] S. Roettger. The Volume Library. http://schorsch.efi.fh-nuernberg.de/data/volume/. Accessed: September 22, 2015.

[26] B. Schlei. A new computational framework for 2D shape-enclosing contours. *Image and Vision Computing*, 27(6):637–647, May 2009. doi:10.1016/j.imavis.2008.06.014.

[27] D. Siedhoff, F. Weichert, P. Libuschewski, and C. Timm. Detection and classification of nano-objects in biosensor data. In *Proc. 6th Int. Workshop on Microscopic Image Analysis with Applications in Biology MIAAB 2011*, Heidelberg, Germany, September 2011.

[28] Z. Wang, J. K. Min, and G. Xiong. Robotics-driven printing of curved 3D structures for manufacturing cardiac therapeutic devices. In *Proc. IEEE Int. Conf. Robotics and Biomimetics ROBIO 2015*, pages 2318–2323, December 2015. doi:10.1109/ROBIO.2015.7419120.

[29] D. Wu, H. Tian, G. Hao, et al. Design and realization of an interactive medical images three dimension visualization system. In *Proc. 3rd Int. Conf. Biomedical Engineering and Informatics BMEI 2010*, volume 1, pages 189–193, Oct 2010. doi:10.1109/BMEI.2010.5639435.