Applied Medical Informatics
Original Research **Vol. 28, No. 1 / 2011, pp: 9-15**

Automatic Contour Extraction from 2D Image

Panagiotis GIOANNIS*

Technological Educational Institute of Athens, Department of Physiotherapy, Ag. Spyridonos Str., 12210 Aigaleo, Greece.
E-mail: p_gioannis@yahoo.com

* Author to whom correspondence should be addressed; Tel.: +302105758482.

Received: 15 December 2010 /Accepted: 21 February / Published online: 7 March 2011

Abstract:

Aim: To develop a method for automatic contour extraction from a 2D image. *Material and Method:* The method is divided in two basic parts where the user initially chooses the starting point and the threshold. Finally the method is applied to computed tomography of bone images. *Results:* An interesting method is developed which can lead to a successful boundary extraction of 2D images. Specifically data extracted from a computed tomography images can be used for 2D bone reconstruction. *Conclusions:* We believe that such an algorithm or part of it can be applied on several other applications for shape feature extraction in medical image analysis and generally at computer graphics.

Keywords: Contour extraction; Medical image; 2D image; Computed tomography.

Introduction

There is a plethora of algorithms for contour extraction from medical images. They are quite useful tools which can be applied at computed tomography (CT), magnetic resonance (MR), X-ray, angiography, ultrasound and other medical data. Except medical images those methods can be used to several other applications of image processing. Some of those methods can be filters [1], snake (or deformable) models [2,3] or a variation of it B-snakes [4], and chain code first introduced by Freeman [5] and commonly used by Bribiesca [6]. Also contour information can be extracted through segmentation methods [7,8]. There exists plethora of ideas based on variation of those methods. An example is the recent work introduced by Li [9]. Nowadays contour extraction still is an attractive problem for a research. Therefore solutions come through new methods such as computational geometry [10], neural networks [11], a Bayesian inference approach [12] and so on.

Most of these algorithms have to deal with: accuracy of results, the automatism procedure, time complexity, noise of data and efficiency. However, in our opinion the most important feature has to do with the accuracy of the extracted information compared with the needs of the real problem. For example we would like to extract the appropriate inner/outer contours from computer tomography (CT) slices of a residual human limb to build 3D bone model which will be the basis for development of a prosthetic finite element (FE) model [13]. This appears to be a quite difficult technical problem since we are trying to approximate a real model. We will deal with the first part which has to do with the automatic extraction of the inner/outer contours. Therefore we suggest a simple and accurate algorithm which can extract the appropriate contours as an input for the 3D model.

Proposed Method

The method is separated in three continuous parts. Firstly, the boundaries of the inner/outer contours will be extracted within a specific threshold starting from a seed point which is manually picked by the user. Then the boundaries will be thinning leaving only a one pixel width boundary. Also at this step the non-boundary pixels will be removed. Finally, boundary points will be chosen within an appropriate step after a B-spline interpolation.

Step I: Boundary Extraction

The method is based on flood-fill algorithm [14]. The input will have three parameters:

- an initial seed point which will be chosen manually by the user. The seed point must be between inner and outer boundaries but not within any islet with threshold close to boundary threshold.
- a threshold which will defined by the user. However the threshold must be similar for all the CT slices.
- a second point named within.inner.boundary which will be chosen manually by the user. The within.inner.boundary must be within the inner contour. If the user needs to extract more than one inner boundary (a bone with two inner boundaries) then s/he must click within the other inner boundaries.

The idea of flood-fill algorithm is simple with a propagation starting from the seed point and preceding until reaching the boundaries (out of threshold). The algorithm using a 4-connected area is:

```
function Fill (column, row)
{
 current.color = GetPixelcolor (column, row)

 if (current.color != paint.color && current.color <= threshold)
   {
    PaintPoint (column, row, paint.color)
    Fill (column+1, row)
    Fill (column-1, row)
    Fill (column, row+1)
    Fill (column, row-1)
   }
 else if (current.color != paint.color && current.color > threshold)
    Add.Point.Boundary (column, row)
 }
```

Using the above algorithm the region between inner and outer contours will be painted temporarily with ***paint.color***. The painted region is within the given threshold and the points out of threshold will be the boundary points. The problem is that the boundaries will have a thickness more than one point. Also islets with boundary threshold between inner and outer contour will appear on results (Figure 1). All this will be solved at the second step.

Here we use a recursive algorithm which may cause stack overflow because of the memory used. However, nowadays computers memory has increased enough. But we suggest similar algorithms based on queue data structure [15, 16] as an alternative way in case of memory problem.

Step II: Removing islets, Thinning boundary, B-spline

At this step we will proceed with improving the results given from the flood-fill algorithm. Firstly we will remove the islets with boundary threshold which appears between inner and outer contours. Then we will extract one point thick boundary and finally we will fit a B-spline curve on the boundary points.

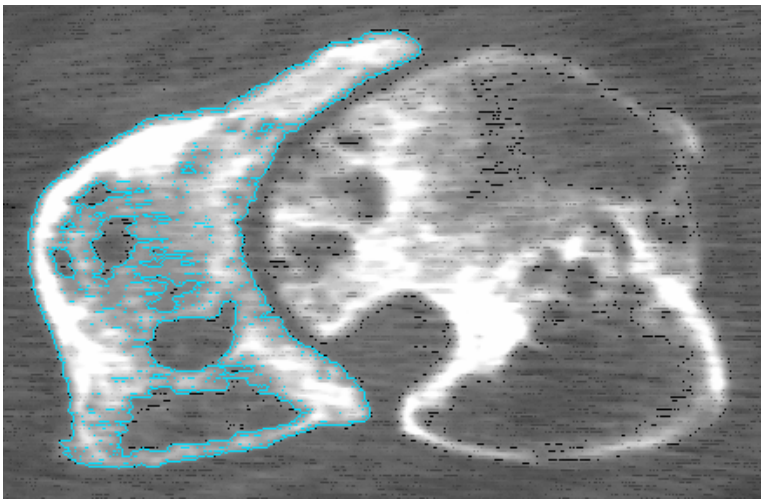

Figure 1. The results of Step I where is shown the extracted boundaries and the islets

Removing Islets

Since the area between inner and outer boundaries is not always clean it is possible to have small regions with the same threshold as the boundaries. That garbage's are produced from the first step and are not necessary for the final results. We suggest the following algorithm overall previously extracted points to remove as much garbage as we can:

Let's suppose that the boundary point extracted from the fist step has value 1 (Point(column,row)=1) and the rest value 0 (Point(column,row)=0).

```
function Check (column, row)
{
  /*for a given  column and row the following variables are calculated.
 all: number of all active neighbors: min=0, max=8
 vertical: number of all active vertical neighbors: min=0, max=8
 diagonal: number of all active diagonal neighbors: min=0, max=8*/

 flag = 0

 if (all==1 || all==0)
    {
     Point(column, row) = 0
     return
    }
 else if (vertical==1 && diagonal==1)
   {
  if (Point(column+1,row-1)==1 && Point(column,row-1)==1) flag=1
  else if (Point(column-1,row-1)==1 && Point(column,row-1)==1) flag=1
  else if (Point(column+1,row+1)==1 && Point(column,row+1)==1) flag=1
  else if (Point(column-1,row+1)==1 && Point(column,row+1)==1) flag=1
  else if (Point(column-1,row+1)==1 && Point(column-1,row)==1) flag=1
  else if (Point(column-1,row-1)==1 && Point(column-1,row)==1) flag=1
  else if (Point(column+1,row+1)==1 && Point(column+1,row)==1) flag=1
  else if (Point(column+1,row-1)==1 && Point(column+1,row)==1) flag=1

  if (flag==1)
     {
      Point(column, row) = 0
      return
     }
   }
}
```

Thinning boundary

At that phase we will continue with the extraction of one point width boundaries (Figure 2).

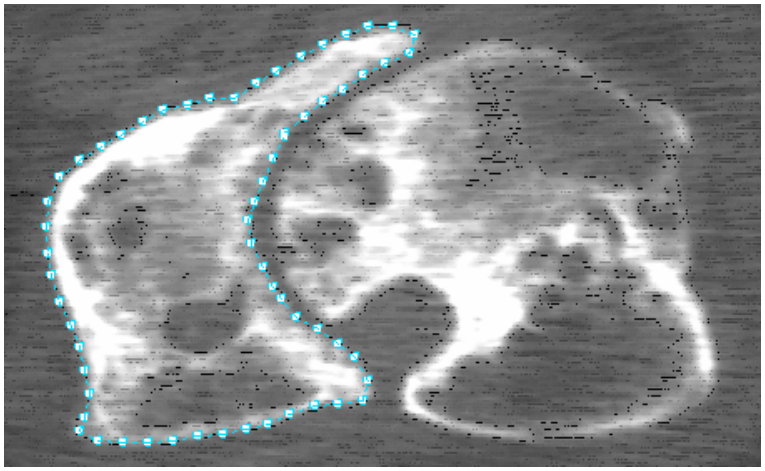

Figure 2. The results of Step II where is shown a sampled one point thick outer boundary

Firstly the outer boundary is produced and then the inner boundary (or boundaries). The starting point of outer boundary (column, row) can be easily chosen by selecting the point with the maximum coordinates. The boundary point is inserted at the end of list through the function Insert.point.list(column, row, flag). A node of the boundary list has: point coordinates, the number of current active neighbors of the point (flag) and a pointer which point to the next node. The pointer TailList refers always to the last node of the list. Also the function DeletePoint() is used to delete the last point of the list and set the pointer TailList to the previous one.

```
function exact_boundary
{
 flag = 0
 column.start = column
 row.start = row
 do{ /*all 8-neighbors combinations*/
  if (Point(column+1,row)==1)
      {
      column = column + 1
      flag = flag + 1
      Point(column, row) = 0
      Insert.point.list (column, row, flag)
      }
  else if (Point(column,row+1)==1)
      {
      row = row + 1
      flag = flag + 1
      Insert.point.list (column, row, flag)
      }
  else if (Point(column-1,row)==1)
      {
      column = column - 1
      flag = flag + 1
      Insert.point.list (column, row, flag)
      }
  else if (Point(column,row-1)==1)
      {
      row = row - 1
      flag = flag + 1
      Insert.point.list (column, row, flag)
      }
  else if (Point(column+1,row-1)==1)
```

```
        {
        column = column + 1
        row = row - 1
        flag = flag + 1
        Insert.point.list (column, row, flag)
        }
    else if (Point(column-1,row+1)==1)
        {
        column = column - 1
        row = row + 1
        flag = flag + 1
        Insert.point.list (column, row, flag)
        }
    else if (Point(column+1,row+1)==1)
        {
        column = column + 1
        row = row + 1
        flag = flag + 1
        Insert.point.list (column, row, flag)
        }
    else if (Point(column-1,row-1)==1)
        {
        column = column - 1
        row = row - 1
        flag = flag + 1
        Insert.point.list (column, row, flag)
        }
    else
      {
      if (flag>9)
       {/*Here is the stopping criteria, when the boundary closed*/
       if (column==column.start && row==row.start) break;
       else if (column==column.start   && row==row.start+1) break;
       else if (column==column.start   && row==row.start-1) break;
       else if (column==column.start+1 && row==row.start)   break;
       else if (column==column.start-1 && row==row.start)   break;
       else if (column==column.start+1 && row==row.start+1) break;
       else if (column==column.start-1 && row==row.start-1) break;
       else if (column==column.start+1 && row==row.start-1) break;
       else if (column==column.start-1 && row==row.start+1) break;
       }
     /*If the point does not have neighbors then delete it and return to the
         previous point of the list*/
       DeletePoint()
     column = TailList->column
     row = TailList->row
     flag = TailList->flag

     }

   }while (1) /*end of infinite while*/
}
```

The same function exact_boundary() is used to calculate the inner boundary with the starting point calculated using the initial picked point within.inner.boundary which is within the inner boundary. Moving outward from the point within.inner.boundary the first boundary point we meet is the starting point for extracting the inner boundary. Of course before the inner contour extraction the points of outer boundary are deleted from the list.

The inner contour can be extracted with an alternative way by choosing from the beginning, after calculating the outer boundary, a seed point within the inner boundary. However, we decide to choose that way so the data to meet the same threshold criteria since the area within inner contour has a different threshold scale from the area between inner and outer contour. Also it is not necessary to pass Step I.

Finally, the user can define the sample rate of the extracted boundaries points. That if is not

necessary to keep one-point step boundary for a further processing.

B-spline

At that phase B-spline interpolation has been made a using De Boor's algorithm [17] over the extracted boundary pointed. Then the boundaries are resampled to generate equally spaced points. B-spline will guarantee the same degree and smoothness (cubic and twice continuously differentiable) over all the slices, which will help on the 3D modeling.

Conclusions

The method that we presented is automatic (except the three initial parameters chosen initially by the user manual) for extraction of inner/outer contours of 2D images. It is a practical and accurate method which at some way resembles the Magic Wand Tool of Adobe Photoshop software. However, with Adobe Photoshop software we can take as result only a processed image and not the boundary pixels. Therefore a further process is needed. Here we suggest completed package for a boundary extraction.

Obviously, some parts of the methods need improvement in terms of computational complexity which we suggest for future research. However, in this article we care more for the accuracy of the extracted data which can be used for a further process on building real models.

The method can be successfully applied for extraction of boundaries as an input of a 3D model, specifically as described above for computer tomography images. However, we believe that it can be easily applied to several other applications not only medical but generally computer graphics.

References

1. Canny J. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 1986;8(6):679-714.
2. Kass M, Witkin A, Terzopoulos D. Snakes: Active contour models. International Journal of Computer Vision 1988;1(4):321-31.
3. Menet S, Saint-Marc P, Medioni G. B-Snake: Implementation and Application to Stereo. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics; 1990 November 4-9; Los Angeles, CA. p. 720-726.
4. McInerney T, Terzopoulos D. Deformable models in medical image analysis: a survey. Medical Image Analysis 1996;1(2):91-108.
5. Freeman H. On the encoding of arbitrary geometric configurations. IRE Trans. Electron. Comput 1961;10:260-8.
6. Bribiesca E. A new chain code. Pattern Recognition 1999;32:235-51.
7. Pal NR, Pal SK. A Review on Image Segmentation Techniques. Pattern Recognition 1993;26(9):1277-94.
8. Pham DL, Xu C, Prince JL. Current Methods in Medical Image Segmentation. Annual Review of Biomedical Engineering 2000;2:315-7.
9. Li J. Image Contour Extraction Based on Ant Colony Algorithm and B-snake. In Proceedings of the 6th International Conference on Advanced Intelligent Computing Theories and Applications: Intelligent Computing; 2010 August 18-21; Changsha, China. p. 197-204.
10. Tejada P, Qi X, Jiang M. Computational Geometry of Contour Extraction. In Proceedings of the 21st Canadian Conference on Computational Geometry; 2009 August 17-19; Vancouver, BC, Canada. p. 25-28.
11. Zhiheng Z, Zhengfang L, Delu Z. Contour Extraction Algorithm Using a Robust Neural Network. In Proceedings of the 7th international conference on Computational Science, Part IV; 2007 May 27 - 30; Beijing, China. p. 283-290.

12. Dong X, Zheng G. Automatic extraction of femur contours from calibrated x-ray images: A Bayesian inference approach; In Proceedings of the 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro; 2008 May 14-17; Paris, France. p. 57-60.
13. Saxena R, Zachariah SG., Sanders JE. Processing computer tomography bone data for prosthetic finite element modeling: A technical note. J Rehabil Res Dev 2002;39(5):609-14.
14. Hearn D, Baker MP. Computer Graphics C Version. 2d ed.: Prentice Hall; 1996.
15. Albert TA, Slaaf DW. A Rapid Regional Filling Technique for Complex Binary Images. Computer and Graphics 1995;19(4):541-9.
16. Silvela J, Portillo J. Breadth-First Search and Its Application to Image Processing Problems. IEEE Transactions on Image Processing 2001;10(8):1194-9.
17. De Boor C. A Practical Guide to Splines. Revised Edition New York: Springer-Verlag; 2001.