Moore Neighbor Tracing

Release 0.00

David Doria

August 1, 2011

Rensselaer Polytechnic Institute

Abstract

This document presents an implementation of Moore Neighbor Tracing - an algorithm to find an ordered outline of a blob or contour in an image.

An excellent tutorial on Moore Neighbor Tracing is provided here: http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html

The code is available here: https://github.com/daviddoria/MooreTracing

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3308]

Contents

1 Introduction **1**

2 Algorithm **2**

3 Implementation **2**

4 Example **3**

5 Code Snippet **3**

1 Introduction

This document presents an implementation of Moore Neighbor Tracing - an algorithm to find an ordered outline of a blob in an image. The algorithm also works for images already containing contours (non-filled

blobs) in that it assigns an ordering to the contour pixels.

An excellent tutorial on Moore Neighbor Tracing is provided here:

http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html

This particular implementation is designed to be used with a "label image", typically the output of a pipeline consisting of one of many filters that produces an itkLabelMap (e.g. itkBinaryImageToLabelMapFilter) followed by a itkLabelMapToLabelImageFilter. Therefore, since we are looking to order the pixels in an already-known-to-be closed blob or contour, the termination criterion that has been implemented is returning to the starting pixel. Moore Neighbor Tracing can also be used with other stopping criteria to order pixels of open contours.

The code is available here: https://github.com/daviddoria/MooreTracing

2 Algorithm

The algorithm proceeds as follows:

Initialization:

1. Locate a boundary pixel. This can be found simply using a raster scan of the image and looking for the first non-background pixel.

Iterate:

1. Start on a boundary pixel. This can be found simply using a raster scan of the image and looking for the first non-background pixel.
2. Record the pixel from which you "entered" the contour pixel.
3. Traverse the 8-connected neighbors (consistently either clockwise or counter-clockwise), looking for another non-background pixel. The direction of the contour will be the same as the choice of the neighborhood traversal direction (either clockwise or counterclockwise). We have arbitrarily chosen counter-clockwise traversal in this implementation.

Termination:

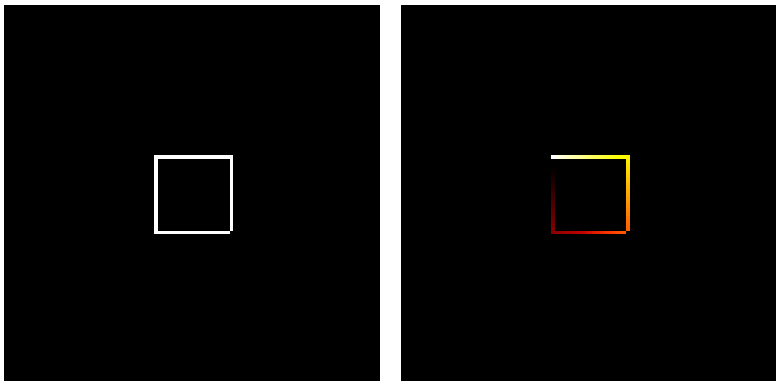
1. Terminate when the initial pixel is reached.

3 Implementation

This implementation only traces the first contour found in an image. If the image contains more than one contour, the user could remove the pixels found in the first contour and then call MooreTrace until there are no more contours.

4 Example

Figure 1 shows the computed ordering of the pixels on a square. The tracing started in the top left corner and proceeded counter-clockwise around the square.



(a) Input image.

(b) Output image showing the ordering of pixels (black to white (i.e. Hot colormap)).

Figure 1: Example of Moore Neighbor Tracing.

5 Code Snippet

```
// Create an image
ImageType::Pointer image = .... obtain an image from somewhere ....

// Trace the contour
std::vector< itk::Index<2> > path = MooreTrace(image);

// Output the ordered contour pixels
for(unsigned int i = 0; i < path.size(); ++i)
  {
  std::cout << path[i] << std::endl;
  }
```