

# Find tunnel 'center line'?

Asked 10 years, 6 months ago    Active 7 years, 8 months ago    Viewed 4k times

▲

10

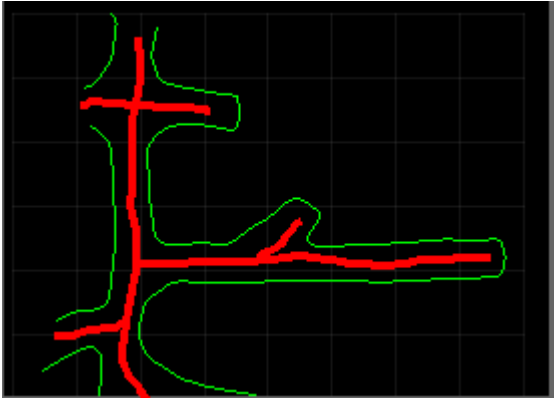
▼

★

12

🕒

I have some map files consisting of 'polylines' (each line is just a list of vertices) representing tunnels, and I want to try and find the tunnel 'center line' (shown, roughly, in red below).



I've had some success in the past using [Delaunay triangulation](#) but I'd like to avoid that method as it does not (in general) allow for easy/frequent modification of my map data.


Any ideas on how I might be able to do this?

[algorithm](#)   [image](#)   [language-agnostic](#)   [graphics](#)   [geometry](#)

Share   Improve this question

Follow

edited Feb 28 '11 at 2:42

 **Dr. belisarius**  
59.1k   13   107   187

asked Oct 21 '10 at 1:24

 **sje397**  
38.9k   7   79   102

- Your diagram needs more freehand circles! Joking aside, I assume the complaint about Delaunay triangulation is based on the runtime? How precise does your solution need to be? Can you improve runtime to an acceptable level by simply dropping proximate vertices? They're likely to have a better answer for this question over at the GIS stackexchange site. Unfortunately, since it's beta, I can't vote to move the question over there, but I strongly encourage you to re-ask it there. [gis.stackexchange.com](#)

– Paul McMillan Oct 21 '10 at 1:34
- Thanks Paul. I'll try there. I have some very large maps, and very limited runtime resources. My current requirements don't call for extreme accuracy. I want to be able to modify my map and not recalculate the whole triangulation, and the implementations I've seen of Delaunay allowing both removal and addition of points are either not very efficient or extremely complicated. I just have a feeling that there is a much easier way.

– sje397 Oct 21 '10 at 1:40
- 1   Question at GIS.se: [gis.stackexchange.com/questions/2775/find-tunnel-center-line](#) – sje397 Oct 21 '10 at 2:23
- I don't know how extensive your modifications are, but is recalculating just a subset of the data an option? There are good spatial partitioning schemes that would let you select only nearby stuff to work with.

– Paul McMillan Oct 21 '10 at 2:25
- 1   @belisarius: I want to end up with a line (in the maths sense) - i.e. the walls are infinitely thin lines (vectors) and I want to get an infinitely thin center line as output. – sje397 Oct 21 '10 at 13:07



## An "algorithm" that works well with localized data changes.



### The critic's view



#### [The Good](#)

The nice part is that it uses a mixture of image processing and graph operations available in most libraries, may be parallelized easily, is reasonable fast, may be tuned to use a relatively small memory footprint and doesn't have to be recalculated outside the modified area if you store the intermediate results.

#### [The Bad](#)

I wrote "algorithm", in quotes, just because I developed it and surely is not robust enough to cope with pathological cases. If your graph has a lot of cycles you may end up with some phantom lines. More on this and examples later.

#### [And The Ugly](#)

The ugly part is that you need to be able to flood fill the map, which is not always possible. I posted a comment a few days ago asking if your graphs can be flood filled, but didn't receive an answer. So I decided to post it anyway.

### The Sketch

The idea is:

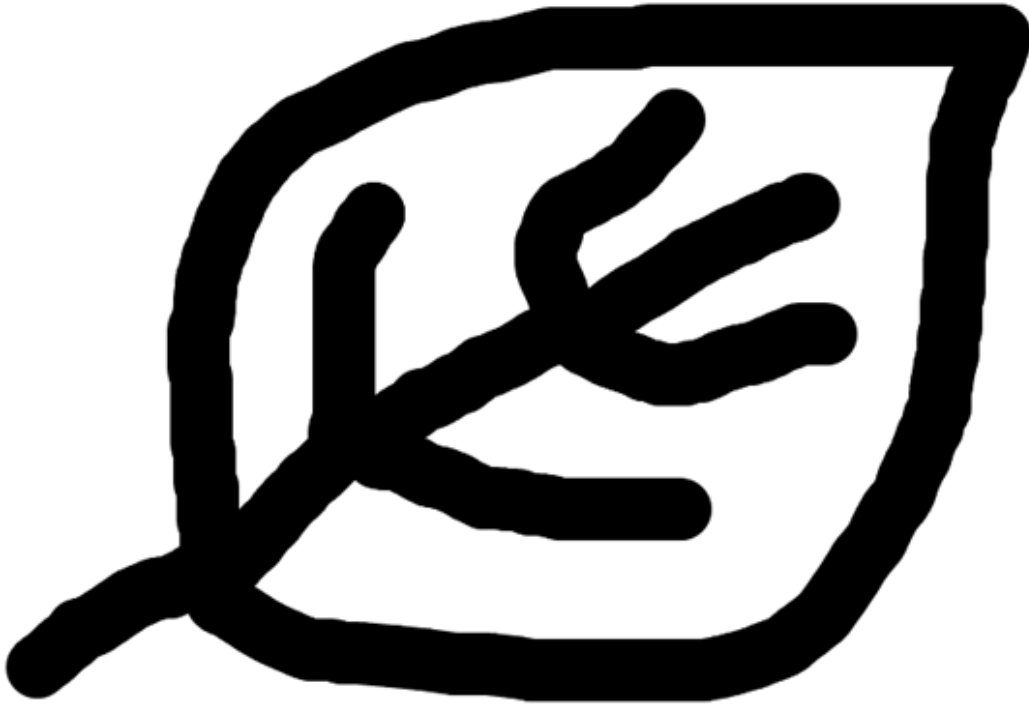
1. Use image processing to get a fine line of pixels representing the center path
2. Partition the image in chunks commensurated to the tunnel thinnest passages
3. At each partition, represent a point at the "center of mass" of the contained pixels
4. Use those pixels to represent the Vertices of a Graph
5. Add Edges to the Graph based on a "near neighbour" policy
6. Remove spurious small cycles in the induced Graph
7. End- The remaining Edges represent your desired path

The parallelization opportunity arises from the fact that the partitions may be computed in standalone processes, and the resulting graph may be partitioned to find the small cycles that need to be removed. These factors also allow to reduce the memory needed by serializing instead of doing calcs in parallel, but I didn't go through this.

### The Plot

I'll no provide pseudocode, as the difficult part is just that not covered by your libraries. Instead of pseudocode I'll post the images resulting from the successive steps. I wrote the program in [Mathematica](#), and I can post it if is of some service to you.

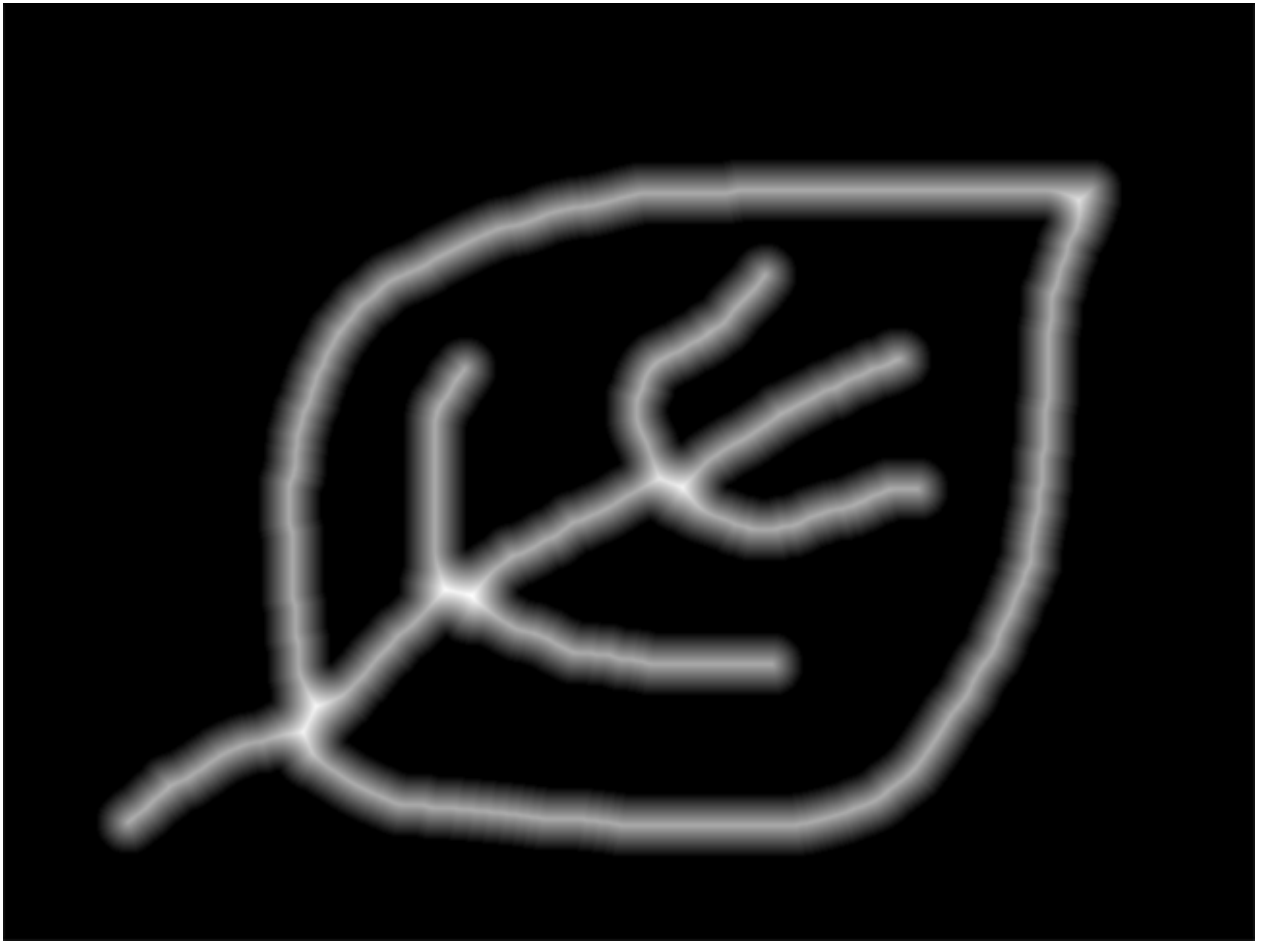
#### **A- Start with a nice flood filled tunnel image**



#### **B- Apply a Distance Transformation**

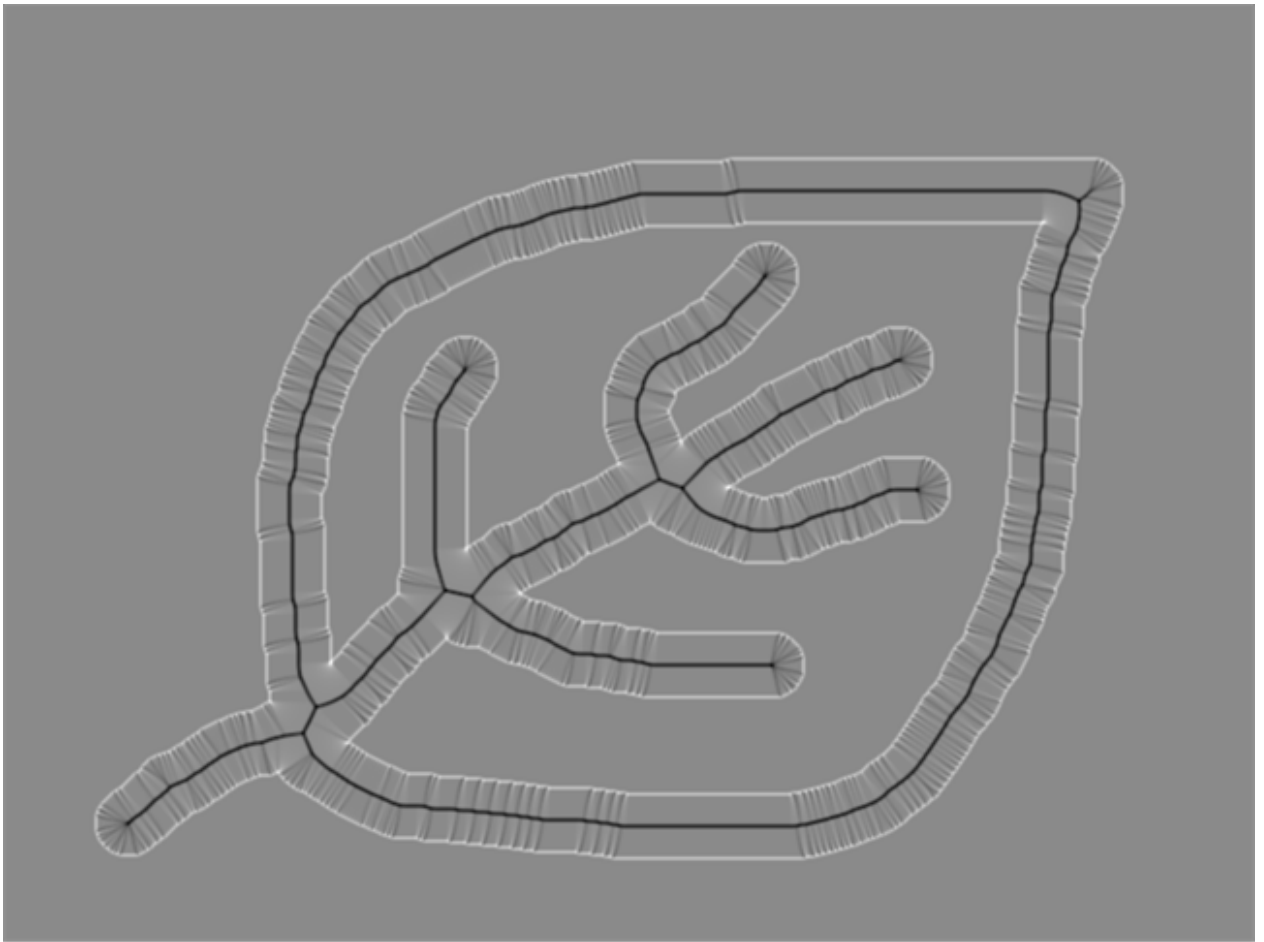
The [Distance Transformation](#) gives the distance transform of image, where the value of each pixel is replaced by its distance to the nearest background pixel.

You can see that our desired path is the Local Maxima within the tunnel



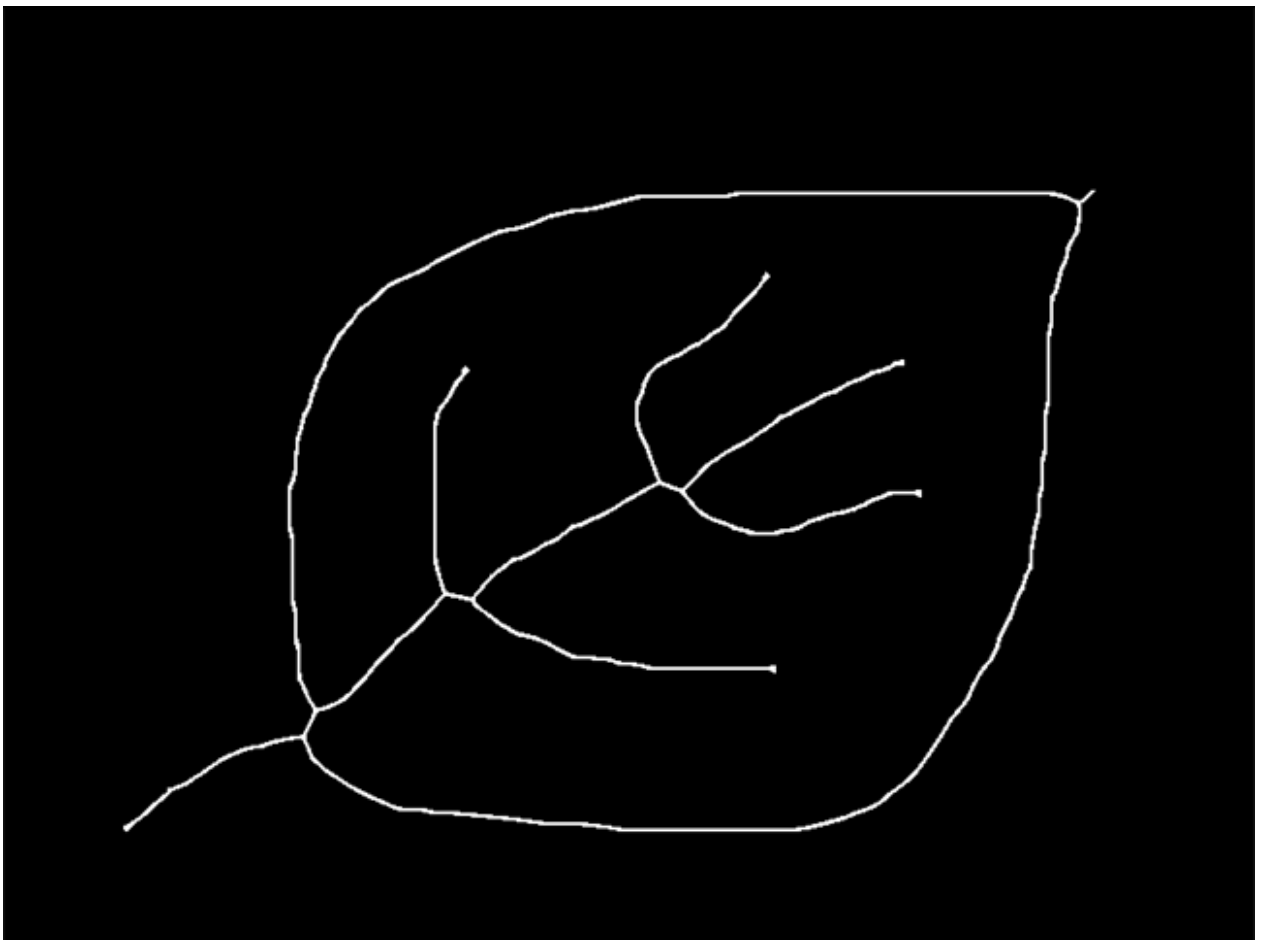
### C- Convolve the image with an appropriate kernel

The selected kernel is a [Laplacian-of-Gaussian kernel](#) of pixel radius 2. It has the magic property of enhancing the gray level edges, as you can see below.



#### D- Cutoff gray levels and Binarize the image

To get a nice view of the center line!

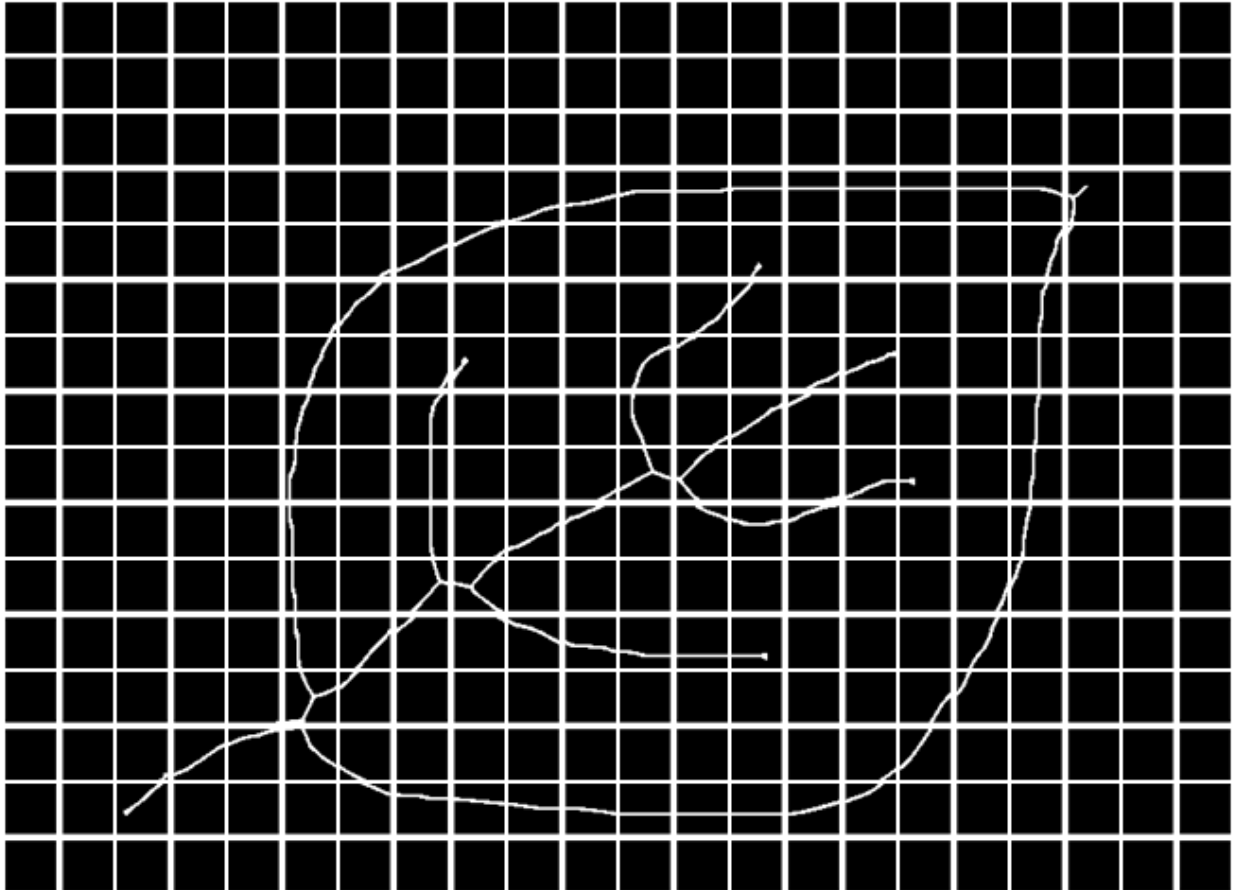


## Comment

Perhaps that is enough for you, as you may know how to transform a thin line to an approximate piecewise segments sequence. As that is not the case for me, I continued this path to get the desired segments.

## E- Image Partition

Here is when some advantages of the algorithm show up: you may start using parallel processing or decide to process each segment at a time. You may also compare the resulting segments with the previous run and re-use the previous results



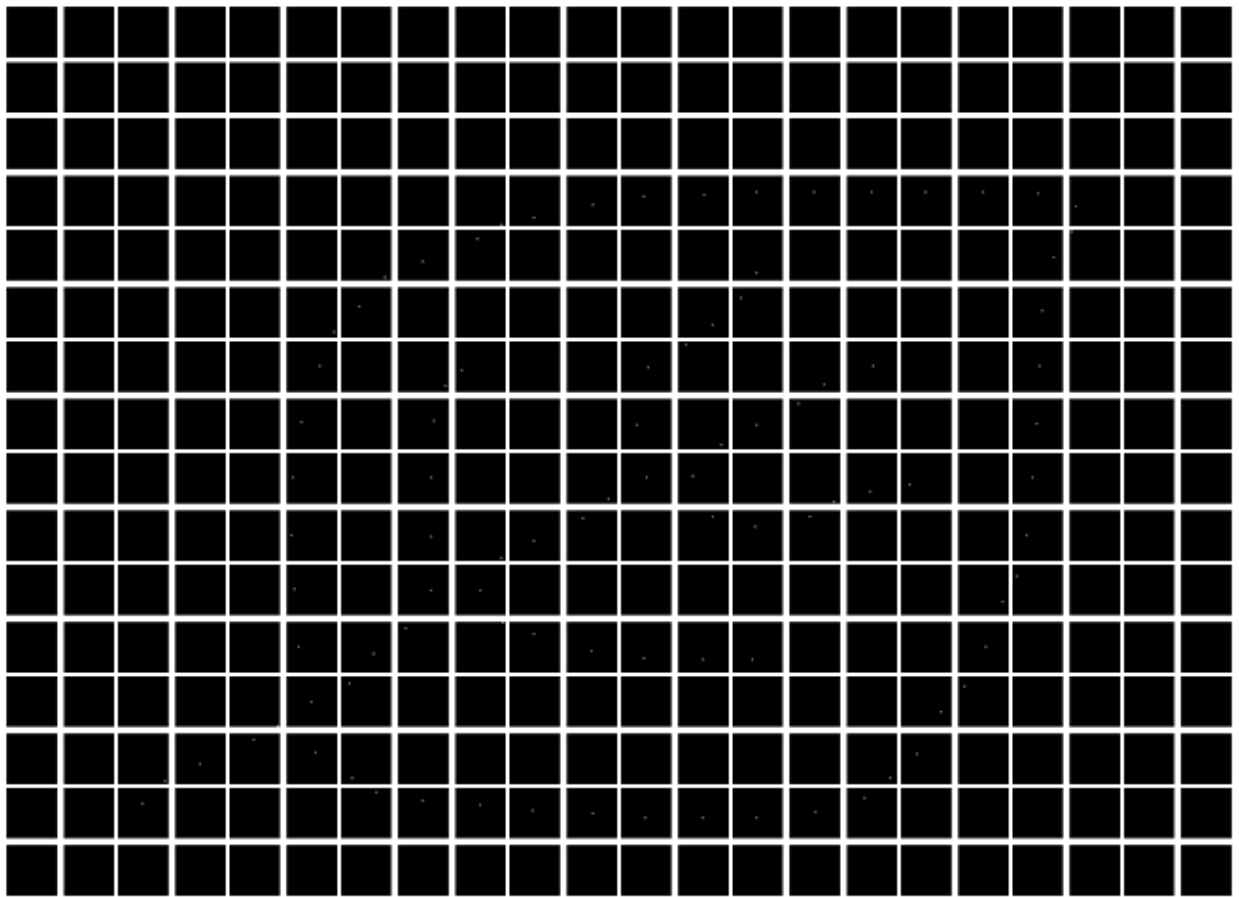
## F- Center of Mass detection

All the white points in each sub-image are replaced by only one point at the center of mass

$$X_{CM} = (\sum_{i \in Points} X_i) / NumPoints$$

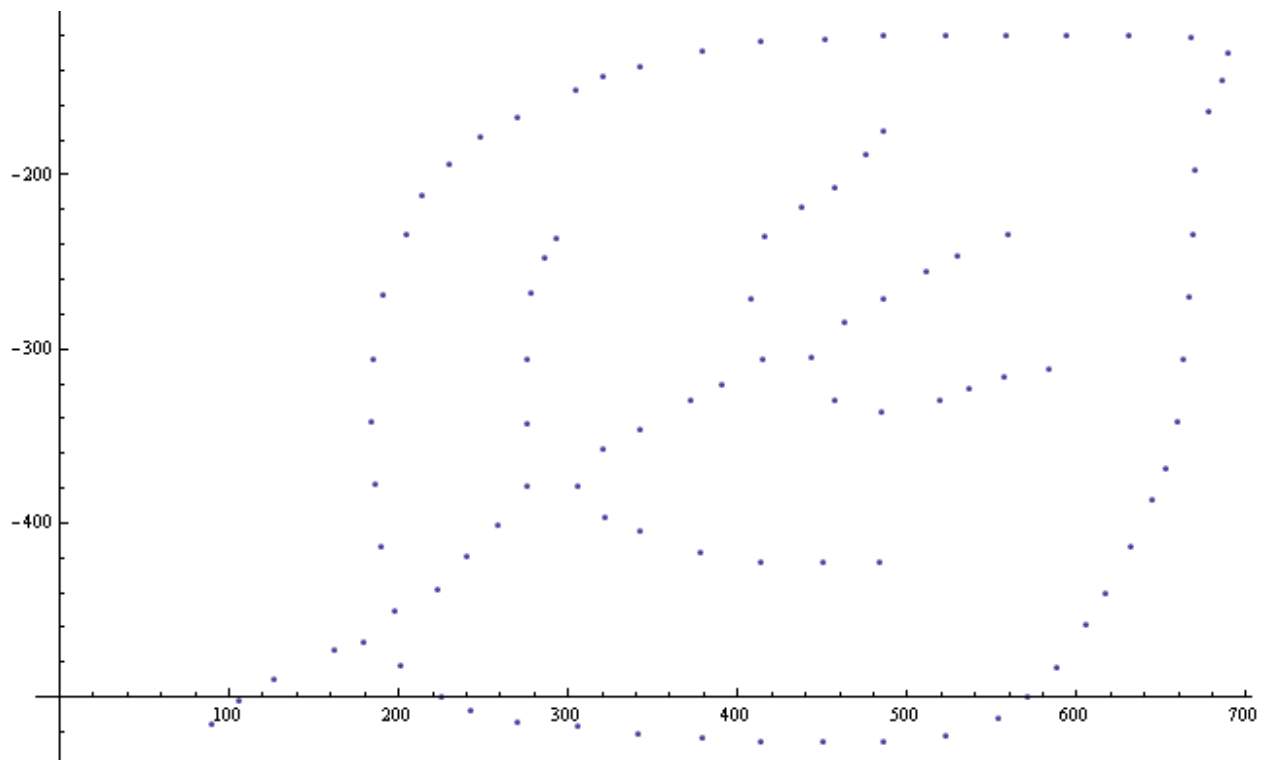
$$Y_{CM} = (\sum_{i \in Points} Y_i) / NumPoints$$

The white pixels are difficult to see (asymptotically difficult with param "a" age), but there they are.



### G- Graph setup from Vertices

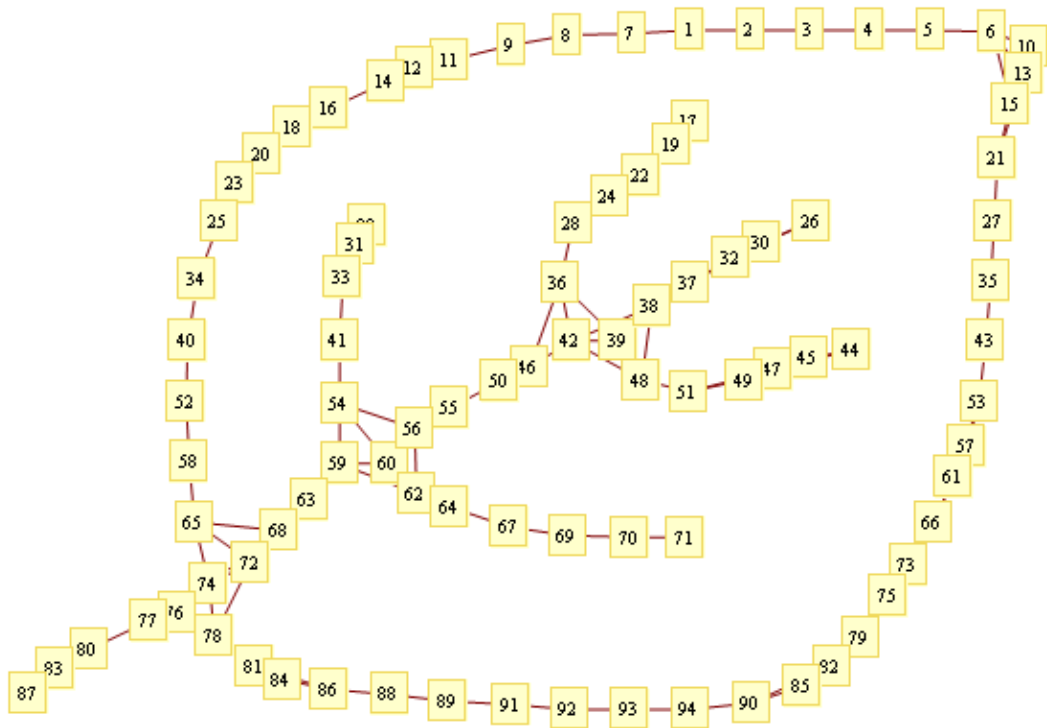
Form a Graph using the selected points as Vertex. Still no Edges.



### H- select Candidate Edges

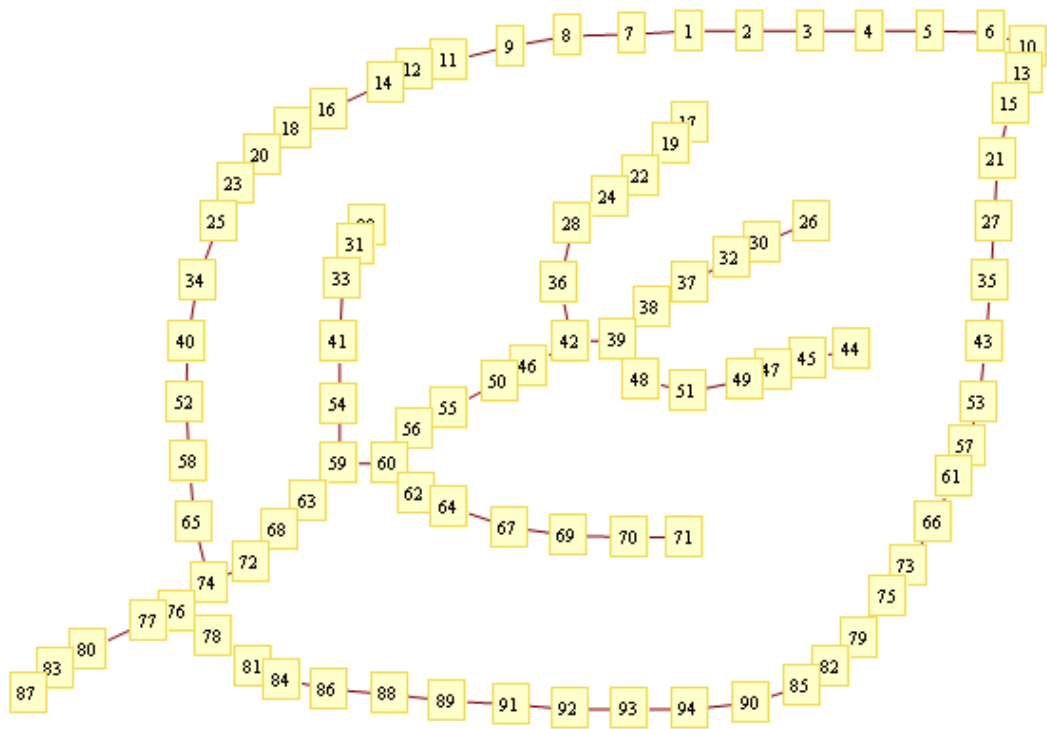
Using the Euclidean Distance between points, select candidate edges. A cutoff is used to select an appropriate set of Edges. Here we are using 1.5 the subimage size.

As you can see the resulting Graph have a few small cycles that we are going to remove in the next step.



## H- Remove Small Cycles

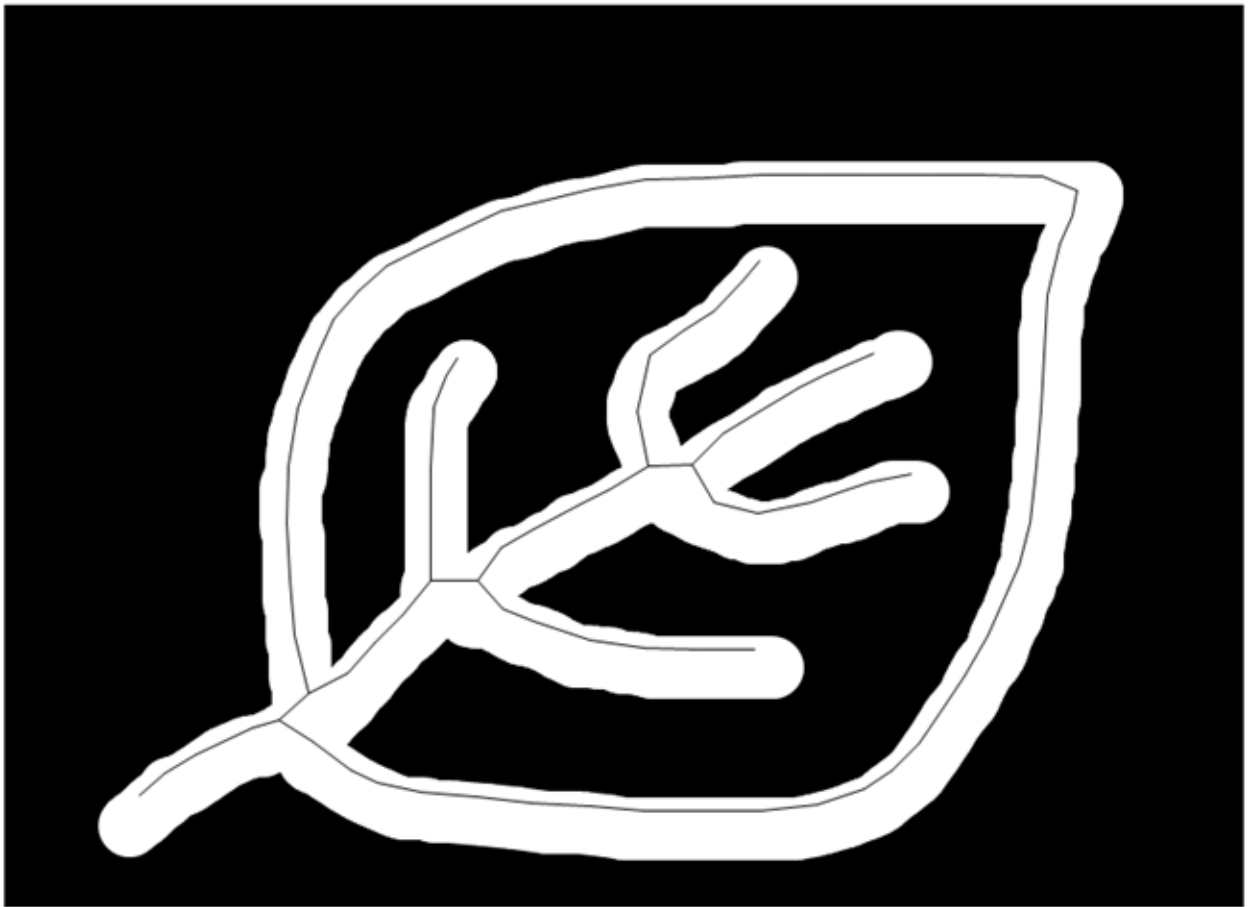
Using a Cycle detection routine we remove the small cycles up to a certain length. The cutoff length depends on a few parms and you should figure it empirically for your graphs family





## I- That's it!

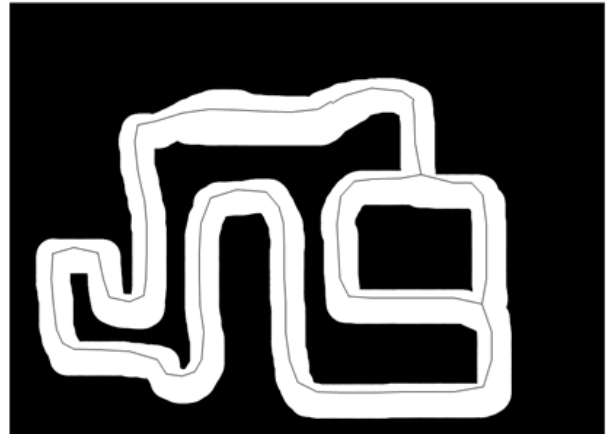
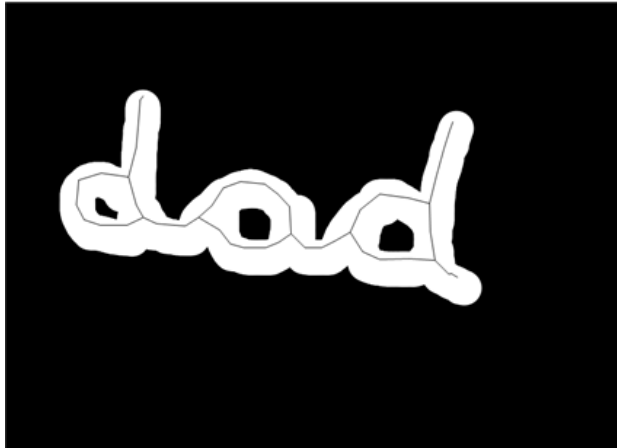
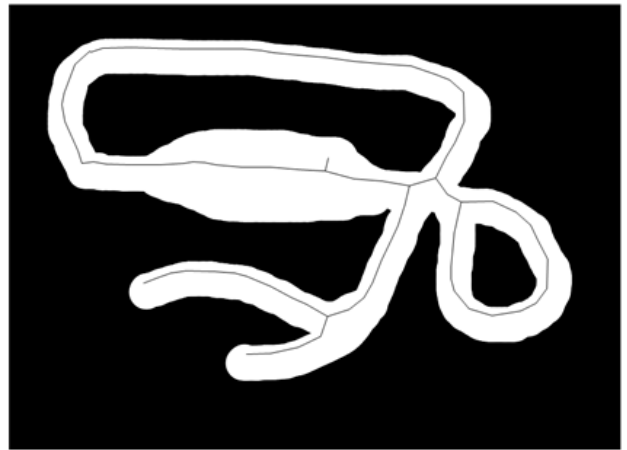
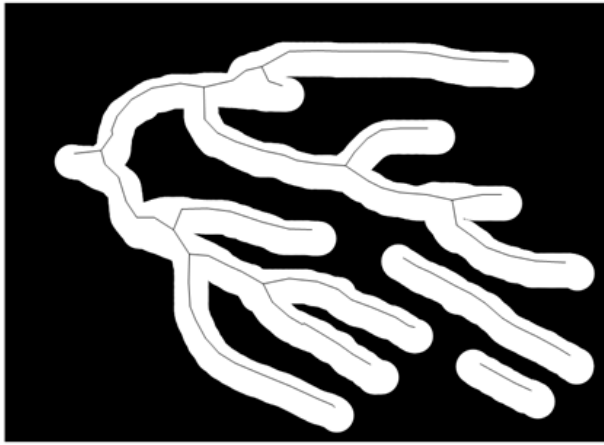
You can see that the resulting center line is shifted a little bit upwards. The reason is that I'm superimposing images of different type in Mathematica ... and I gave up trying to convince the program to do what I want :)



## A Few Shots

As I did the testing, I collected a few images. They are probably the most un-tunnelish things in the world, but my Tunnels-101 went astray.

Anyway, here they are. Remember that I have a displacement of a few pixels upwards ...



HTH !

.

## Update

Just in case you have access to [Mathematica 8](#) (I got it today) there is a new function **Thinning**. Just look:

In[37]:= a =



In[39]:= b = Thinning@ColorNegate@a

Out[39]=



In[40]:= ImageAdd[a, b]

Out[40]=



Share Improve this answer

edited Jan 10 '11 at 1:53

answered Oct 25 '10 at 9:38

Follow



[Dr. belisarius](#)

59.1k 13 107 187

- 1 Awesome answer. Apologies for not answering re flood-filling - the truth is I'm not really sure if that's a possibility. I will give it some thought. There's heaps here to think about. Thank you very much for putting in so much effort. – [sje397](#) Oct 25 '10 at 9:56
- 1 +1. Impressive! And I have to imagine that a local re-fill wouldn't be too difficult after an initial annotation of tunnel widths per vertex (computed after the first fill). It shouldn't be too hard to cut out a region around the changes and apply something like this just to the cutout. – [Rex Kerr](#) Oct 25 '10 at 19:47

@Rex Kerr The algorithms for deleting edges and merging graphs are not difficult. In fact I am already merging one point at a time. If the modified region has a simple geometric expression (rectangle, circle ...) the whole thing is easy. I think I'll post an example if the floodfill is viable. – [Dr. belisarius](#) Oct 26 '10 at 0:31

@Rex Kerr To be more specific: Re-merging a modified area needs two geometric objects ..1) The modified area and 2) It's frontier in the original map. From the frontier you get the graph vertices to create candidate proximity edges with the new zone – [Dr. belisarius](#) Oct 26 '10 at 1:01

@belisarius - Agreed. I don't think it will be a trivial change, but I don't see any huge conceptual

problems with it if one can make certain assumptions about vertex spacing and such. Anyway, again, impressive! – [Rex Kerr](#) Oct 26 '10 at 15:35

4

This is a pretty classic skeletonization problem; there are lots of algorithms available. Some algorithms work in principle on outline contours, but since almost everyone uses them on images, I'm not sure how available such things will be. Anyway, if you can just plot and fill the sewer outlines and then use a skeletonization algorithm, you could get something close to the midline (within pixel resolution).



Then you could walk along those lines and do a binary search with circles until you hit at least two separate line segments (three if you're at a branch point). The midpoint of the two spots you first hit, or the center of a circle touching the three points you first hit, is a good estimate of the center.

Share Improve this answer Follow

answered Oct 21 '10 at 5:42



[Rex Kerr](#)

162k 24 307 402

- 1 A skeleton on it's own will be accurate enough. The problem is that the implementations I've found haven't dealt well with frequently changing map data (i.e. localized changes in very large maps). – [sje397](#) Oct 21 '10 at 13:32
- 1 Most algorithms won't propagate local changes very far anyway--why not just cut out the local part of the map, re-skeletonize, and heal up any differences at the end (e.g. linear interpolation between the two answers over some distance)? – [Rex Kerr](#) Oct 21 '10 at 14:22

3

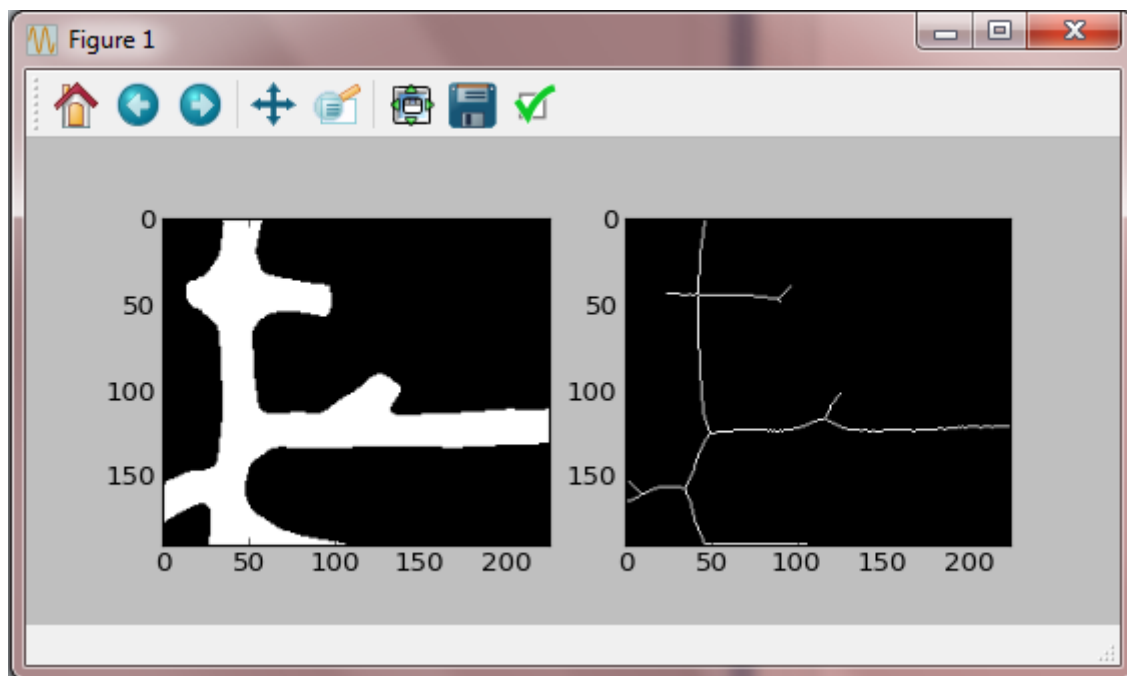
Well in [Python](#) using package [skimage](#) it is an easy task as follows.



```
import pylab as pl
from skimage import morphology as mp

tun = 1-pl.imread('tunnel.png')[...,0]      #your tunnel image
sk1 = mp.medial_axis(tun)                    #skeleton

pl.subplot(121)
pl.imshow(tun,cmap=pl.cm.gray)
pl.subplot(122)
pl.imshow(sk1,cmap=pl.cm.gray)
pl.show()
```



Share Improve this answer Follow

answered Aug 6 '13 at 12:01



[Developer](#)

7,044 7 43 53