Flux driven medial curve extraction

Release 1.0

Xavier Mellado Esteban[1], Ignacio Larrabide[1], Monica Hernandez[2] and Alejandro Frangi[1]

August 3, 2007

[1]Computational Imaging Lab, Department of Technology, Pompeu Fabra University,
Psg. Circunval·lació 8 - Barcelona, Spain
ignacio.larrabide@upf.edu
[2]Aragon Institute of Engineering Research (I3A), University of Zaragoza, Spain

Abstract

In this document it is described the implementation of the flux driven automatic centerline extraction algorithm proposed by Bouix *et al.* in 2004. This is based on a skeletonisation algorithm initially proposed by Siddiqi *et al.* in 2002, using properties of an average outward flux measure to distinguish skeletal points from non-skeletal ones. This information is combined with a topology preserving thinning procedure to obtain the final result. This implementation combines this skeletonisation algorithm with other techniques as described in the paper of Bouix *et al.* to produce an ITK filter that generates as output the skeleton, as a binary object represented in an image, of the input surface, represented as a distance transform image. In this work is described this medial curve extraction procedure following the ITK philosophy.

Contents

1 Introduction

In this work is described a method that computes the centerline from the distance transform representation of a surface. Flux driven medial curve [1] is a method for extracting centerlines based on a recently developed skeletonisation algorithm proposed by Siddiqi *et al.* [4].

The approach is based on an algorithm whose theoretical properties have been thoroughly justified [4, 2], when other topological thinning methods presented in the literature use heuristics for the selection of anchor points. Also, the algorithms developed here lend themselves to the use of quite standard numerical implementations, finding all centerline paths in volumetric tubular structures having arbitrary topology. In most cases, methods presented in the literature are designed to find a single centerline path at a time and have numerical implementations that are more complex. The implemented approach is fully automatic and requires no user interaction. Several of the other methods do require the user to select at least the end points of a particular centerline path.

This algorithm uses the Average Outward Flux (AOF) of the input image. The AOF is computed using the following equation:

$$AOF(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{26} \langle \mathbf{n}_i, \nabla D(\mathbf{x}_i) \rangle \tag{1}$$

where $\mathbf{x}_i$ is one of the 26-neighbors at $\mathbf{x}$, $\mathbf{n}_i$ is the outward normal at $\mathbf{x}_i$ of the unit sphere centered at $\mathbf{x}$ and D is the Euclidean distance function.

The strategy followed to extract the centerline path is to thin the medial surface to obtain a structure composed of only one curve (the medial curve) and then prune the result to obtain well centered paths. The process used to obtain the medial curve is described in Algorithm 1. In order to make the algorithm more computationally efficient, an ordered heap structure is used. We call *simple* points[1], those points that, when removed, do not change the topology of the object [1]. Such a point has the property that its removal: i) does not create a hole, ii) does not create a cavity and iii) does not disconnect a connected component. The classification of a simple point is a specific case of a more general categorization of a simple point x in a cubic lattice first introduced by [3]. This categorization is based on two numbers:

- C^*: the number of 26-connected components 26-adjacent to x in $O \cap N_{26}^*$.
- $\overline{C}$: the number of 6-connected components 6-adjacent to x in $\overline{O} \cap N_{18}^*$.

Here O is a 26-connected object, N_{26}^* is the 26-neighborhood of x without x and N_{18} is the 18-neighborhood of x including x. Simple (and hence removable) points can be identified by the conditions $C^* = 1$ and $\overline{C} = 1$. The AOF *threshold* parameter is the average outward flux threshold at which end points are preserved in the extraction of the medial curve.

The code that implements this algorithm is in the class `MedialCurveImageFilter`. This class actually requires two inputs:

- The surface from which the centerline will be computed. This information should be provided in its distance map representation. This is a real valued function, and the input point (pixel) type should be `float` at least.
- The average outward flux: it is computed as described in the work of Bouix *et al.* [1] in class `AverageOutwardFluxImageFilter`, also provided. The input pixel type should be `float` at least as AOF is a real valued function.

[1] In the case of the numerical implementation, a point corresponds to a pixel.

Algorithm 1 Algorithm 1 - Medial curve extraction.

Require: A surface implicitly represented by its signed distance map S representation, the AOF of this distance map at each point, AOF *threshold*.

Ensure: The centerline represented as a binary image.

```
for each point x do
  if x is simple then
    OrdHeap.insert(x) // ordered by −D(x) as the sorting key for insertion;
  end if
end for
while OrdHeap.size > 0 do
  x = OrdHeap.pop()
  if x is simple then
    if (x is an end point of a 3D curve) and (AOF(x)< threshold) then
      mark x as a medial curve (end) point;
    else
      remove x;
      for all neighbors y of x do
        if y is simple then
          insert x in OrdHeap an ordered by −D(x) as the sorting key for insertion;
        end if
      end for
    end if
  end if
end while
```

The algorithm output is a binary image containing the centerline of the input structure. Because of this, the output is only allowed to be of type `unsigned char`.

2 Proposed classes and implementation

The classes proposed are:

- `itk::itk::AverageOutwardFluxImageFilter` : computes the AOF of a distance map representing the surface. In this case we are interested in computing the AOF of the input distance map surface representation.
- `itk::itk::MedialCurveImageFilter` : topology preserving thinning procedure that iteratively removes simple elements from the image preserving the number of connected components and avoiding the creation of cavities.

2.1 itk::AverageOutwardFluxImageFilter

A special class for the computation of AOF was implemented. This class implements Eq. (1) in order to compute the AOF for every image pixel. The main class features are presented below.

```
template< class TInputImage,
      class TOutputPixelType = float,
      class TInputVectorPixelType = ::itk::CovariantVector<TOutputPixelType,
                      ::itk::GetImageDimension<TInputImage>::ImageDimension> >
class ITK_EXPORT AverageOutwardFluxImageFilter:
    public ImageToImageFilter<TInputImage, ::itk::Image<TOutputPixelType,
                   ::itk::GetImageDimension<TInputImage>::ImageDimension> >
{
    public:

        typedef typename itk::Image< TInputVectorPixelType,
                            TInputImage::ImageDimension > TInputVectorImage;
        typedef typename itk::Image< TOutputPixelType,
                               TInputImage::ImageDimension > TOutputImage;
[...]

#ifdef ITK_USE_CONCEPT_CHECKING
    /** Begin concept checking */
    itkConceptMacro(SameDimensionCheck,
      (Concept::SameDimension<InputImageDimension, OutputImageDimension>));
    itkConceptMacro(InputVectorIsReallyAVectorCheck,
      (Concept::HasValueType<TInputVectorPixelType>));
    itkConceptMacro(InputVectorIsFloatingPointCheck,
      (Concept::IsFloatingPoint<TInputVectorPixelType::ValueType>));
    itkConceptMacro(InputVectorIsSameDimensionInputImageCheck,
      (Concept::SameDimension<TInputVectorPixelType::SizeType,InputImageDimension>));
    itkConceptMacro(OutputIsFloatingPointCheck,
      (Concept::IsFloatingPoint<TOutputPixelType>));
    /** End concept checking */
#endif
    virtual void SetGradientImage(InputVectorConstPointerType gradient)
    {this->ProcessObject::SetNthInput
              ( 1, const_cast< TInputVectorImage * >(gradient.GetPointer()) );}

    virtual InputVectorConstPointerType GetGradientImage()
    {return ( static_cast< TInputVectorImage *>
             (this->ProcessObject::GetInput(1)) );}

    void GenerateData();

  protected:

    AverageOutwardFluxImageFilter();

    virtual ~AverageOutwardFluxImageFilter();
[...]
};
```

The class template definition is used to ensure consistency of sizes and types between both inputs and the output. The ITK concept checking macros are used for: ensuring that

1. input and output images have the same dimension;
2. input vector image is actually a vector image;
3. input vector image is of floating point precision at least;
4. input vector image is of the same dimension that the input image and to ensure that the output is flouting point precision at least.

2.2 itk::MedialCurveImageFilter

The main algorithm is implemented using a set of protected methods providing some basic and also important information about image pixels. First, a threshold is applied over the object and this information is used as initial data.

```
template< class TInputImage,
      class TAverageOutwardFluxPixelType = float,
      class TOutputPixelType = unsigned char>
class ITK_EXPORT MedialCurveImageFilter:
    public ImageToImageFilter<TInputImage, ::itk::Image<TOutputPixelType,
    ::itk::GetImageDimension<TInputImage>::ImageDimension> >
{
    public:
[...]
        typedef typename itk::Image< TAverageOutwardFluxPixelType,
                    TInputImage::ImageDimension > TAverageOutwardFluxFImage;
        typedef typename itk::Image< TOutputPixelType,
                    TInputImage::ImageDimension > TOutputImage;

    typedef typename TOutputImage::IndexType OutputIndexType;
[...]
    typedef std::vector<Pixel> HeapContainer;
    typedef std::priority_queue<Pixel, HeapContainer, Greater > HeapType;

[...]

    virtual void SetAverageOutwardFluxImage(AOFConstPointerType aofImage)
    {this->ProcessObject::SetNthInput
         ( 1, const_cast< TAverageOutwardFluxFImage * >(aofImage.GetPointer()) );}

    virtual AOFConstPointerType GetAverageOutwardFluxImage()
    {return ( static_cast< TAverageOutwardFluxFImage *>
         (this->ProcessObject::GetInput(1)) );}

#ifdef ITK_USE_CONCEPT_CHECKING
```

```
    /** Begin concept checking */
    itkConceptMacro(SameDimensionCheck,
        (Concept::SameDimension<InputImageDimension, OutputImageDimension>));
    itkConceptMacro(AOFIsFloatingPointCheck,
        (Concept::IsFloatingPoint<TAverageOutwardFluxPixelType>));
    /** End concept checking */
#endif

[...]
  protected:

    MedialCurveImageFilter();
    virtual ~MedialCurveImageFilter();
    bool IsBoundary( OutputIndexType p );
    bool IsIntSimple( OutputIndexType p );
    bool IsExtSimple( OutputIndexType p );
    bool IsEnd( OutputIndexType p );
[...]

};
```

In this case, two inputs are needed by the filter. The first one is the distance map representation of the input surface and the second one is the AOF computed over the distance map. Concept checking macros are used for ensuring input surface and output images have the same dimension and that the average outward flux image, set as second input, is float type.

The method `bool IsBoundary(OutputIndexType p )` returns true if `p` belongs to the object and at least one of its 26 neighbors belongs to background, meaning `p` is exactly at image boundaries. The method `bool IsIntSimple( OutputIndexType p )` returns true if deletion from the object changes local object topology in the 27 neighborhood. An object pixel is simple for object's topology if its deletion from the object does not change the topology in the pixel neighborhood. The algorithm proceeds finding the first 26 connected pixels to the object pixel being visited. A flood fill algorithm goes through the adjacent object pixels different from `p` using 26 connectivity. Finally, if the number of flooded pixels is equal to the number of object pixels-1, and local topology does not change, meaning that the pixel is simple for the object.

The method `bool IsExtSimple( OutputIndexType p )` returns true if its deletion from the object changes local background topology in the 18 neighborhood. An object pixel is simple for the background topology if its deletion from the object does not change background's topology in the pixel neighborhood. The algorithm proceeds finding the first pixel in the background. A flood fill algorithm goes through the adjacent background pixels, different from `p`, using the 6 connectivity in the 18 neighborhood. Finally, if the number of flooded pixels is equal to the number of 6 connected background pixels-1, and the local background topology does not change, the pixel is simple for the background.

The method `bool IsEnd(OutputIndexType p )` returns true if the pixel has less than two object neighbors. That is, `p` is a medial axis end point if in 26-neighborhood there is only one foreground pixel.

Image size (cylinder)	True centerline mean distance	Image size (torus)	True centerline mean distance
$10 \times 40 \times 10$	0.00446911	$30 \times 30 \times 5$	0.043751
$20 \times 80 \times 20$	0.00107213	$60 \times 60 \times 10$	0.014036
$40 \times 160 \times 40$	0.000244575	$120\times 120 \times 20$	0.00458682
$80 \times 320 \times 80$	7.94051e-005	$240\times 240 \times 40$	0.00209226

Table 1: Image size vs. centerline mean distance

3 Tests

Test code for these classes is provided `itkMedialCurveTest.cpp` file. This application receives as input the distance map representation of a surface. The steps followed to compute the centerline are:

1. load the distance map image,
2. apply a light smoothing on the distance map image[2] and compute its gradient,
3. compute the average outward flux using `itk::AverageOutwardFluxImageFilter`,
4. extract the centerline using `itk::MedialCurveImageFilter`.

In order to test the accuracy of the method with respect to the definition of the image, a specific test was performed. In this case, two different synthetic geometries, for which the exact centerline is easily obtainable, where used: a straight tube (length 4, diameter 1) and a semi-torus (240 degrees, curvature radius 7 and tube diameter 1). For both geometries different refinements of the underlying 3D image where generated. For the resulting centerlines was computed the average distance to the real centerline. Results are presented in Table 1.

The output is provided as a binary image containing the centerline of the input structure. This algorithm provides a robust way for extracting centerlines from complex objects that also accounts for bifurcations.

In Figure 1 are presented the results with the corresponding input surfaces from which they where obtained.

4 Conclusion

This paper describes a set of ITK classes that implement the method for centerline extraction proposed in [1]. This implementation is based in computing the average outward flux at every point of the image and subsequently use this information in a robust topology preserving thinning algorithm. As it can be observed in Table 1 and Figure 1, accuracy of the obtained centerline is highly dependent on image resolution, so for more precise results a more refined distance map should be used. It can also be observed (from the algorithm description) that the computational time depends linearly on the image size, as AOF and posterior topology preserving thinning are computed pixel by pixel. In the case of 3D images this value grows very fast. In contrast, very good approximations of centerlines are obtained even for grosse definition of the image.

A Accompanying data

With this code 6 different input data sets are provided:

[2] This pre-processing is performed to improve the quality of the derivatives.

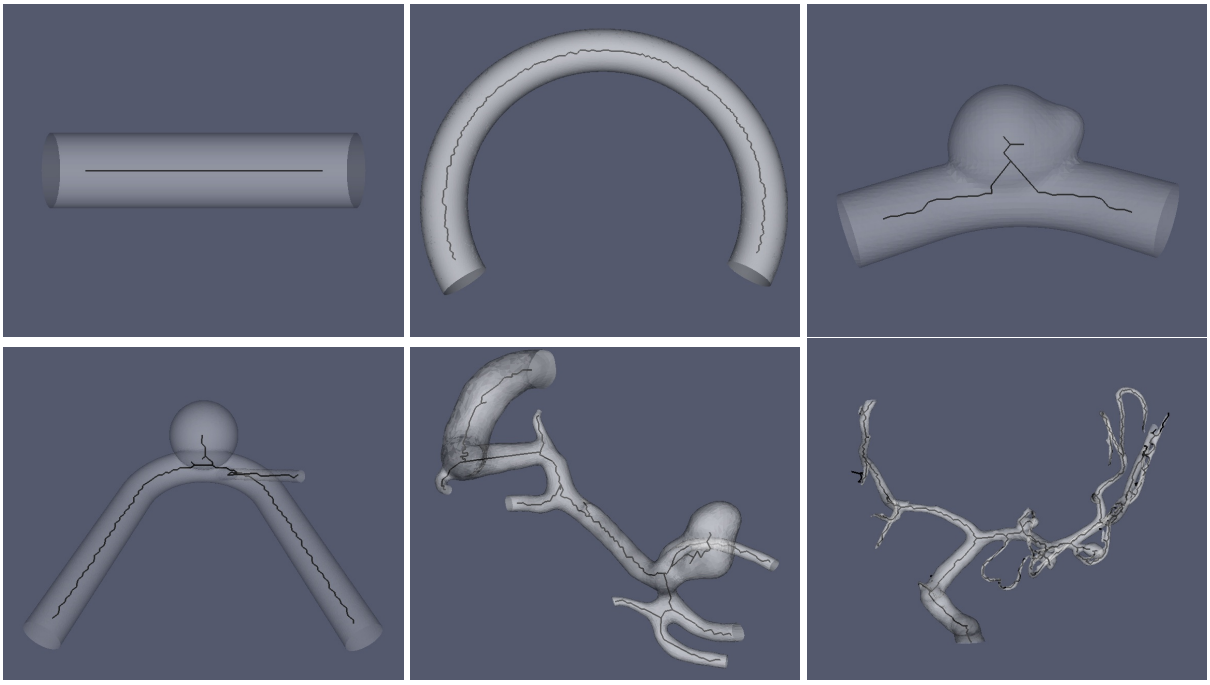

Figure 1: Input geometries and output centerlines for the provided data.

- `cylinder.vtk` : a simple cylinder geometry,
- `torus.vtk` : a torus geometry,
- `aneu_1.vtk, aneu_2.vtk` : two different synthetic artery geometries with aneurysms,
- `aneu_3.vtk, aneu_4.vtk` : two different real artery geometries with aneurysms.

B Acknowledgments

The authors would like to acknowledge to J. R. Cebral, PhD from the George Mason University (Virginia, USA), C. Putman, MD from the Inova Fairfax Hospital (Virginia, USA) and to R. Barrena, MD from Zaragoza University (Spain), for providing the 3DRA and CTA data, respectively, measurements and clinical background. It is also greatly appreciated the help of Prof. José Maria Pozo from the CILab - UPF, for providing the synthetic geometries. The work of X. M. and I. L. were supported by a grant of the Pompeu Fabra University. The work of A. F. F. was supported by the Spanish Ministry of Education & Science under a Ramón y Cajal Research Fellowship. This work was partially generated within the framework of the Integrated Project @neurIST (IST-2005-027703), which is co-financed by the European Commission, and ISCIII IM3 Network (G03/185) funded by Spanish Ministry of Health. And also partially supported by MEC TEC2006-03617, ISCIII FIS2004/40676, and CDTI CENIT-CDTEAM grants. The CILab is part of the ISCIII CIBER-BBN (CB06/01/0061).

References

[1] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. Technical Report SOCS-04.2, School of Computer Science, McGill University, 2004. 1, 1, 1, 4

[2] J. Damon. Global geometry of regions and boundaries via skeletal and medial integrals. Technical report, Department of Mathematics, University of North Carolina at Chapel Hill, 2003. 1

[3] G. Malandain, G. Bertrand, and N. Ayache. Topological segmentation of discrete surfaces. *International Journal of Computer Vision*, 10(2):183–197, 1993. 1

[4] K. Siddiqi, B. B. Kimia, and C. Shu. Geometric shock-capturing eno schemes for sub-pixel interpolation, computation and curve evolution. *Graphical Models and Image Processing*, 59(5):278–301, 1997. 1