Digital Topology

Release 1.00

Julien Lamy [1]

August 23, 2006

[1]lamyj@cc.nih.gov

Abstract

This documents describes a set of classes to integrate digital topology in the Insight Toolkit, with an application to homotopic thinning. The skeletonization filter implemented using our set of classes is twice as fast as `BinaryThinningImageFilter` and can be used on images of any dimension.

Contents

1 Introduction

Digital topology is used in several algorithm, such as skeletonization, topological hull [1] or even the labelling of connected components. However, there is currently no structure in ITK to enable the easy coding of such algorithms. In this paper we describe a set of classes to integrate digital topology in ITK.

These classes are generic with respect to the dimension of the image and to the type of connectivity. They pre-compute as many things as possible to reduce run time, and provide helper classes for usual problems, such as determining the background connectivity from the foreground connectivity.

We have tested the skeletonization algorithm in 2D and compared it to ITK's `BinaryThinningImageFilter`. Our filter is twice as fast as the one currently in ITK. We have also tested it on a 3D image, but could not compare it to another filter, as no 3D thinning is present in ITK.

2 Connectivity

The base of digital topology is the definition of connectivity. To have a connectivity object that is not dependent on the image dimension, we use a definition based on cell decomposition [3]. Let d_i be the dimension of the image. We say that two d_i-cells (*i.e.* pixels or voxels) c_1 and c_2 are (d_i, d_j)-*neighbors* if there exist a d_j-cell ($d_j < d_i$) c such that $c_1 \in \mathrm{St}(c)$ and $c_2 \in \mathrm{St}(c)$, where St is the open star operator.

This notion is implemented by the class `Connectivity` :

```
template<unsigned int VDim, unsigned int VCellDim>
class Connectivity
{
public :
    typedef vnl_vector_fixed<int, VDim> Point;
    typedef unsigned int Offset;

    bool AreNeighbors(Point const & p1, Point const & p2) const;
    bool AreNeighbors(Offset const & o1, Point const & o2) const;
};
```

3 Topological Numbers

Defined by Bertrand and Malandain [2], the topological numbers are an easy way to define the topology of a point p inside an image I, using only p's neighborhood.

This notion is implemented by a function object that counts the neighbors of a point, depending on the connectivity :

```
template< typename TConnectivity,
          typename TNeighborhoodConnectivity =
              typename NeighborhoodConnectivity<TConnectivity>::Type >
class UnitCubeCCCounter
{
public :
    UnitCubeCCCounter();
    ~UnitCubeCCCounter();

    unsigned int operator()() const;

    template<typename Iterator>
    void SetImage(Iterator imageBegin, Iterator imageEnd);
};
```

Here we use a helper class `NeighborhoodConnectivity` to automatically determine the neighborhood to use for the topological numbers computation.

4 Skeletonization Filter

Having introduced a way to characterize the topology of a point, we can easily write a homotopic thinning filter. This filters iteratively removes points that are simple and non-terminal.To guarantee that the resulting

skeleton is correctly centered with respect to the object, the thinning is performed using a distance-ordered removal [4].

The following class performs the homotopic thinning :

```
template<typename TImage, typename TForegroundConnectivity>
class SkeletonizeImageFilter : public InPlaceImageFilter<TImage>
{
public :
    itkGetConstObjectMacro(SimplicityCriterion, Criterion);
    itkSetObjectMacro(SimplicityCriterion, Criterion);

    itkGetConstObjectMacro(TerminalityCriterion, Criterion);
    itkSetObjectMacro(TerminalityCriterion, Criterion);
private :
    typename ImageFunction<TImage, bool >::Pointer m_SimplicityCriterion;
    typename ImageFunction<TImage, bool >::Pointer m_TerminalityCriterion;
};
```

The simplicity and terminality criteria default to usual values, so that the filter can be used easily :

```
Skeletonizer::Pointer skeletonizer = Skeletonizer::New();
skeletonizer->SetInput(image);
skeletonizer->SetOrderingImage(distanceMapFilter->GetOutput());
skeletonizer->Update();
```

5 Results

This is the runtimes in seconds of our method on 2D and 3D images. As ITK's thinning method are only for 2D images, only one timing is shown for the 3D image.

Image	ITK's thinning	Author's method
200×200 2-D image	0.25	0.10
$128 \times 128 \times 90$ 3-D image	N/A	3.82

6 Conclusion

Our digital topology implementation allow to easily code topological algorithms in ITK. We also have shown that our implementation is faster than ITK's current thinning algorithm on 2D images. The files in the archive associated with this report can be used in almost any version of ITK, as they have very limitted dependancies.

References

[1] Zouina Aktouf, Gilles Bertrand, and Laurent Perroton. A three-dimensional holes closing algorithm. *Pattern Recognition Letters*, 23:523–531, 2002. 1

[2] Gilles Bertrand and Grégoire Malandain. A new characterization of three-dimensional simple points. *Pattern Recognition Letters*, 15:169–175, 1994. 3

[3] Vladimir A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics and Image Processing*, 46:141–161, 1989. 2

[4] Chris Pudney. Distance-ordered homotopic thinning : a skeletonization algorithm for 3D digital images. *Computer Vision and Image Understanding*, 72(3):404–413, 1998. 4