IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2014–03–26.
Submitted on 2013–02–13, accepted on 2013–04–16.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2014.74

Extraction of Connected Region Boundary in Multidimensional Images

David Coeurjolly[1], Bertrand Kerautret[2], Jacques-Olivier Lachaud[3]

[1] CNRS, LIRIS UMR5205, Université de Lyon
[2] LORIA, Université de Lorraine
[3] LAMA, Université de Savoie

Communicated by Bertrand Kerautret *Demo edited by* Bertrand Kerautret

Abstract

This paper presents an algorithm to extract the boundary of a connected region(s) using classical topology definitions. From a given adjacency definition, the proposed method is able to extract the boundary of an object in a generic way, independently of the dimension of the digital space.

Source Code

The implementation of the algorithm is available through the DGtal library[1]. The source code and the demonstration are based on a special version of DGTal containing no external dependencies. They are both available on the IPOL web page of this article[2].

Supplementary Material

The DGtalTools[3] project gives several additional tools exploiting the proposed algorithms. These tools are defined to process both 2D and 3D images.

Keywords: Nd connected components, discrete geometry, topology

1 Introduction

The aim of this work is to present an algorithm which extracts the boundary of connected region(s) using classical topology definitions. It can be useful in particular for algorithms which need a discrete contour as input. Such contours can be extracted from grayscale images like the ones displayed in Figure 1.

These digital contours are level set contours in the displayed grey-level image, specified by a threshold parameter (i.e. the level set), a reference point and a maximal distance. In order to describe

[1] *DGtal: Digital Geometry tools and algorithms library*, http://libdgtal.org
[2] https://doi.org/10.5201/ipol.2014.74
[3] https://github.com/DGtal-team/DGtalTools

(a) (b)

Figure 1: Example of contour extraction (b) from source image (a).

in a generic way the object on which the contour is extracted, the user must specify a predicate on volume elements which specifies whether or not a given element belongs to the region(s) of interest. This information is sufficient to extract the interpixel boundaries of this or these region(s). The boundaries are extracted by tracking along the frontier between the region and its complementary. The user may choose between two adjacency definitions for object(s): the *interior* and the *exterior* adjacency. The extracted set of connected surface elements satisfies the choice of adjacency. In dimension 2, the surface elements can be ordered to form contour(s) composed of 4-connected points in the half-integer plane. Predicates on volume elements can be very simple such as the example described in Figure 1: the predicate is just a composition of *thresholds on the image values*.

Complexity note: extracting all the contours for a given predicate takes $O(MN)$ calls to the predicate, if M and N are respectively the width and height of the image. In arbitrary dimension, the computational cost is proportional to the size of the predicate domain (number of grid points). Extracting a single contour given a starting boundary element takes $O(C)$ calls to the predicate, if C is the number of points of this contour.

Implementation note: although the examples and applications are given in 2D and 3D, the surface extraction algorithm is presented and implemented for arbitrary dimension. The applications of Section 3 can be reproduced from the demo version given IPOL web page of this article[4].

2 Algorithm

2.1 Digital Surfaces for Boundary Extraction

Different definitions of digital surfaces can be found in the digital topology domain (refer to the paper by Kong et al. [2] for a survey). A first group of approaches was introduced by Rosenfeld in the 70s [5, 6] which consists in defining a digital surface as a subset S of $\mathbb{Z}^n$ with the property to have $\mathbb{Z}^n \backslash S$ composed of two α-connected (with for instance $\alpha = 4$ or $\alpha = 8$ for the square pixels of an 2D image) components and S is thin (i.e. if any point of S is removed, the preceding property does not hold). Even if such a definition can be used in 2D, it appears more difficult to exploit in higher dimension. A second type of definition considers surfaces as $n-1$ dimensional cubical complexes [3]. This representation is convenient to describe the object but is not adapted to process geometric information of the object boundary. To extract the boundary of a digital surface, we shall use another definition which relies on the set of $n-1$-cells with some specific adjacency (in the same

[4] https://doi.org/10.5201/ipol.2014.74

spirit as Herman [1] or Udupa [7]). Note that we consider here only the implementation for regular grids.

Digital surface as a set of $n-1$-cells. Formally, the elements of a digital space $\mathbb{Z}^n$ are called spels (and often pixels in 2D and voxels in 3D). A surfel is a couple (u, v) of face adjacent spels. A digital surface is a set of surfels. A spel is thus a n-cell in a cellular grid decomposition of the space, while a surfel is clearly some oriented $n-1$-cell which is incident to the two n-cells (see figure 2).

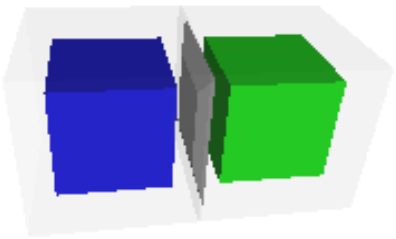


Figure 2: Illustration of a surfel ($n-1$-cell) with its two incident spels (n-cells) in dimension 3.

From the implementation point of view, the set of $n-1$-cells can be obtained with an incidence relation from the given spels. In order to be able to extract object boundaries in a consistent way, we also consider cells with a specific orientation (positive or negative). By convention, spels lying in the interior of the object of interest are given a positive orientation. In the DGtal library framework, the orientation can be specified in the KSpace class (see user documentation[5]):

```
#include "DGtal/helpers/StdDefs.h"
...
using namespace DGtal::Z3i;
...
KSpace K;
// An initial 3D signed  spel defined for instance with positive ↩
   orientation
SCell v = K.sSpel( Point( 0, 0, 0 ), KSpace::POS );
SCell sx = K.sIncident( v, 0, true ); // surfel further along x
```

We can now obtain the digital surface that lies in the boundary of some digital shape $S \subset \mathbb{Z}^n$ as the set of oriented surfels between spels of S and spels not belonging in S. Algebraically, S is the formal sum of its positively oriented spels, and its boundary is obtained by applying the boundary operator on S. Figure 3 illustrates the linear boundary operator applied on the set of spels of the object. By linearity, the operator is applicable spel by spel and when opposite cells appear they cancel each other (see case (c,d) and (h,i)).

Algorithm 1 exploits this definition to extract, given an input shape, the set of surfels that constitutes the shape boundary independently of its dimension. A nD digital Khalimsky space (the class `KhalimskySpaceND`[6] in the DGtal library) is used to represent the cubical grid complex, whose oriented cells are defined as an array of integers (see the paper by Lachaud [4] for more details). In particular, the method `uSpel(Point p)` creates a cell of maximal dimension from a point with coordinates in $\mathbb{Z}^n$. Inversely, the coordinate of a `Cell` can be recovered from a `Cell` with the method `uCoords(Cell c)`.

[5] http://libdgtal.org/doc/nightly/dgtal_cellular_topology.html
[6] http://libdgtal.org/doc/nightly/classDGtal_1_1KhalimskySpaceND.html

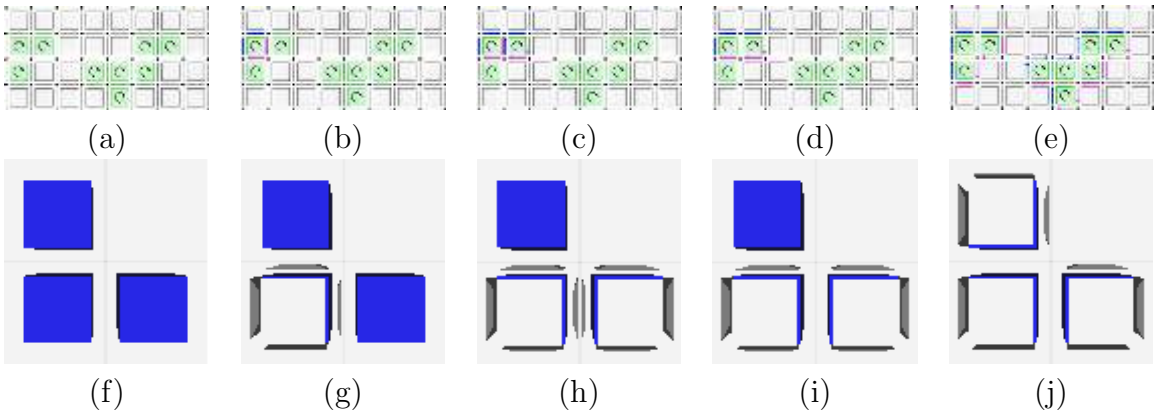


Figure 3: Application of the boundary operator to compute the boundary of a digital shape S, defined either as a 2D set of pixels (images (a-e)) or a 3D set of voxels (images (f-i)).

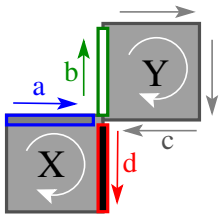
Algorithm 1: Extract the (unstructured) set of surfels that forms the boundary of the digital shape defined by the predicate *Pred*.
(Method `DGtal::Surfaces::detectBoundarySurfels` in the DGtal framework)

```
   input : KhalimskySpaceND K ; // Any Khalimsky space
   input : Point lowerB ; // Lowest point in the space
   input : Point upperB ; // uppermost point in the space
   input : PointPredicate Pred ; // A predicate Point -> bool
   ouput: SurfelSet aBoundary; // The detected set of surfels, i.e. the boundary.
 1 Integer k ; // Current dimension for searching surfels
 2 bool inHere, inFurther;
 3 for k = 0 to K.dimension - 1 do
 4     Cell kLowerB = K.uSpel( lowerB ) ; // lowest spel along k
 5     Cell kUpperB = K.uGetDecr( K.uSpel( upperB ), k ) ; // uppermost spel along k
 6     Cell p = kLowerB ; // current spel
 7     while p in bounds kLowerB and kUpperB do
 8         Cell pNext = K.uGetIncr( p, k) ; // next spel along k
 9         inHere = Pred( K.uCoords(p) );
10         inFurther = Pred( K.uCoords(pNext) );
11         if inHere != inFurther then
               // boundary element, add it to the set
12             aBoundary.insert( K.sIncident( K.signs( p, inHere ), k, true ));
13         p = first spel after p in bounds kLowerB and kUpperB ;
14 return aBoundary ;
```

The boundary operator algorithm is given by the method `DGtal::Surfaces<TKSpace>::detectBoundarySurfels` from the directory `DGtal/topology/helpers`. Images (a,f) of Figure 4 illustrate the result of this method applied on a set of pixels (a) and on a set of voxels (f) by calling the same method `detectBoundarySurfels`.

Once the digital set of surfels is defined, the relation between surfels needs to be determined to transform the digital surface into a graph.

Digital surface as a graph: adding adjacencies between surfels To apply this transformation we have to connect surfels that share $n-2$-cells. The resulting adjacency relation are called *bel adjacencies* in the terminology of Herman [1], Udupa [7] and others. Generally an $n-2$-cell is shared by two $n-1$-cells, except in "cross configuration" which are illustrated in the figure on the right.



The interior bel adjacency makes the choice to connect a to d and c to b while the exterior bel adjacency connects a to b and c to d. This choice has to be made along each possible pair of directions when going nD. In DGtal, it is encoded through the class `SurfelAdjacency`. Images (b,g) in Figure 4 illustrate such possible choices for a given surfel in 2D and 3D. More precisely the green (resp. red) surfel associated to the choice of exterior (resp. interior) adjacency is obtained as follows:

```
...
SurfelAdjacency<Dim>  adjInt( true );  // Interior surfel
SurfelAdjacency<Dim>  adjExt( false ); // Extrior surfel

// surfel is a given surfel boundary.
SurfelNeighborhood<KSpace> sNeighExt;
sNeighExt.init( &ks, &sAdjExt, surfel );
// Axis along which we search for neighbors.
Dimension i = *(ks.sDirs(surfel));

SCell surfelFollowerInt;
SCell surfelFollowerExt;

sNeighInt.getAdjacentOnDigitalSet ( surfelFollowerInt, aSet, i, true);
sNeighExt.getAdjacentOnDigitalSet ( surfelFollowerExt, aSet, i, true);
```

A more "classical topology" way of interpreting adjacencies is to consider an ϵ-offset to the set of cells (see Figure 5). In 2D, an outward ϵ-offset to the boundary cells (or equivalently, the boundary of the Minkowski sum of the set of spels and an ϵ-ball) defines a 1-dimensional surface whose topology corresponds to the exterior adjacency. Similarly, an inward ϵ-offset to the boundary cells defines a 1-dimensional surface whose topology corresponds to the interior adjacency. Unfortunately, this is no more true in 3D. Indeed, an outward ϵ-offset in the great diagonal configuration (two diagonally opposite spels in $2\times2\times2$ cube) connects the surface cells, but they are not connected by the digital exterior adjacencies. However, the inward ϵ-offset to the set of cells still defines the interior adjacency.

2.2 Tracking Digital Surfaces

Once the surfels separating interior spels from exterior spels have been extracted by Algorithm 1, the final step of tracking can be applied. We propose Algorithm 2 which is based on the more generic method `getAdjacentOnPointPredicate`, which only requires a predicate on a digital point. A digital set S is then simply defined as the predicate returning true on points belonging to S. This algorithm is independent of the chosen dimension. This method can be used indifferently for an open

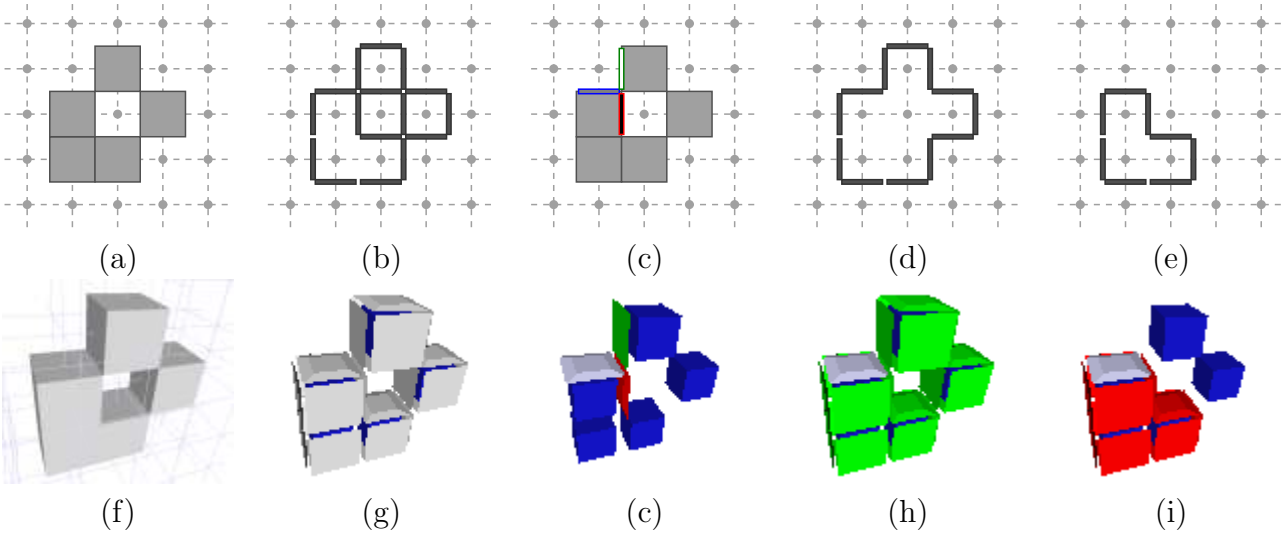


Figure 4: Illustration of the extraction of the set of boundary surfel (b) from the pixel set (a). The bel adjacency (interior in red, and exterior in green) is illustrated starting from the initial surfel in blue (c). Images (d) and (e) illustrate the interior and exterior tracking from the initial blue surfel. The source codes used to generate these illustrations is given in the directory `IllustrationsArticleIPOL` of the `FrechetAndConnectedCompDemo` archive (needs DGtal dependencies with *QGLviewer*).

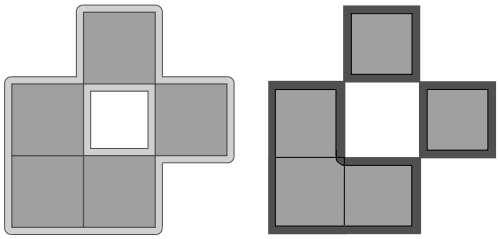


Figure 5: Interpreting digital adjacencies as offsets to the set of cells: (left) outward ϵ-offset corresponding to the exterior adjacency (in light grey), (right) inward ϵ-offset corresponding to the interior adjacency (in dark grey).

or closed surface (with or without boundary). If we know that the surface is closed, the faster variant `trackClosedBoundary` performs the scan by following only direct adjacent surfels.

For the specific aim of extracting an ordered sequence of boundary elements in 2D, we use a variant of Algorithm 2, `track2DBoundary` which constructs the ordered sequence from one direction and takes into account the case of open contour by reversing the sequence and by continuing in the other direction.

2.3 Overall Description

The main steps of the surface extraction algorithm are the following:

1. $B \leftarrow$ detect boundary surfels (Algorithm 1, method `Surfaces::detectBoundarySurfels` in DGtal framework)
2. pick a surfel in B
3. $S \leftarrow$ track the boundary component that contains B (Algorithm 2, method `Surfaces::trackBoundary` in DGtal framework)

Algorithm 2: Tracking a boundary component in nD.
(method `DGtal::Surfaces::trackBoundary` in the DGtal framework)

```
   input : KhalimskySpaceND K ; // Any Khalimsky space
   input : SurfelAdjacency surfelAdj ; // A surfel adjacency
   input : Surfel startS ; // the surfel where the tracking is initiated
   input : PointPredicate Pred ; // A predicate Point -> bool
   ouput: SurfelSet surface ; // The boundary component that contains startS
 1 Surfel b; // the current surfel
 2 Surfel bn; // the neighboring surfel
 3 SurfelNeighborhood SN; // An object that extracts neighboring surfels,
       initialized from K, surfelAdj and startS,
 4 Queue<Surfel> Q // Queue of surfels, the ''head'' of the tracking.
 5 Q.push( startS ) ;
 6 surface.insert( startS ) ;
 7 while Q is not empty do
 8     b = Q.front();
 9     Q.pop();
10     SN.setCurrentSurfel( b ) ; // Position neighborhood around b
11     for All tracking directions trackDir around b do
           // Over a surfel there are n-1 possible axis tracking directions.
           // 1st pass with positive orientation
12         if SN.getAdjacentOnPointPredicate( Out bn, Pred, trackDir, true ) then
13             if surface.find( bn ) == surface.end() // if not found
14             then
15                 surface.insert( bn ) ;
16                 Q.push( bn ) ;
           // 2nd pass with negative orientation
17         if SN.getAdjacentOnPointPredicate(Out bn, Pred, trackDir, false ) then
18             if surface.find( bn ) == surface.end() // if not found
19             then
20                 surface.insert( bn ) ;
21                 Q.push( bn ) ;
22 return surface;
```

4. remove S from B (line 10-13 of Algorithm 3).

5. go back to 2 until B is empty.

These steps are gathered in Algorithm 3, which extracts for each connected component its set of surfels. In the DGtal library the algorithm is given in the method `extractAllConnectedSCell` included in class `Surfaces` (file `DGtal/topology/helpers/Surfaces.h`). This algorithm can process digital objects given in nD, and the resulting sets of surfels are not given in a particular order. To obtain specifically a set of 2D contours (which can be represented as sequences), we just adapt the algorithm by modifying the tracking (line 8 of Algorithm 3) with the specific 2D tracking (variant of Algorithm 2, method `Surfaces::track2DBoundary` in the DGtal library). Such variant of Algorithm 3 is available in DGtal with the method `DGtal::Surfaces::extractAll2DSCellContours`.

Algorithm 3: Given a 2D predicate describing a 2D digital shape implicitly, extracts all boundary components as a vector of 2D contours. Each 2D contour is a sequence of surfels. (method `DGtal::Surfaces::extractAllConnectedSCell` in the DGtal framework)

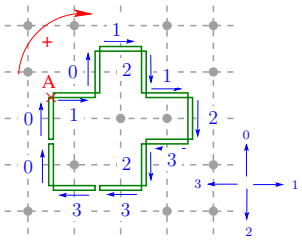
```
   input : KhalimskySpaceND K ; // Any Khalimsky space
   input : SurfelAdjacency surfelAdj ; // A surfel adjacency
   input : PointPredicate Pred ; // A predicate Point -> bool
   ouput: vector< vector<Surfel> > bdryComponents // A vector containing all the
          vector of connected surfels
 1 SurfelSet wholeBoundary ; // The whole detected boundary
 2 wholeBoundary ← detectBoundarySurfels( K, Pred, K.lowerBound(), K.upperBound() ) ;
          // Call to Algorithm 1
 3 while wholeBoundary is not empty do
 4     vector<Surfel> aVector ; // initialize a vector of surfel
 5     Surfel b = first element of the set wholeBoundary;
 6     aVector ← track2DBoundary( K, surfelAdj, b, Pred ) ; // Call Algorithm 2
 7     allBoundaries.push_back( aVector );
       // removing cells from boundary
 8     for int i = 0 to aVector.size() -1 do
 9         wholeBoundary.erase( aVector[i] ) ;
10 return allBoundaries ;
```

3 Examples of Applications

We present two applications of these algorithms, first for 2D contour extraction, and second for 3D connected components extraction.

3.1 Contour Extraction on 2D Grayscale Images

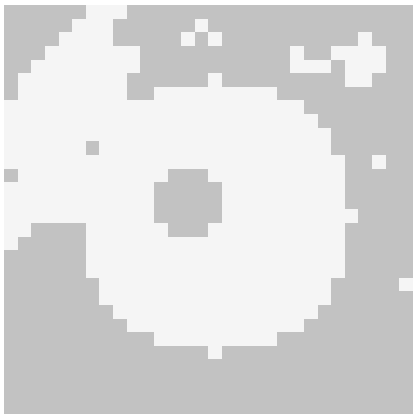
The algorithm of contour extraction was applied on the grayscale images of Figure 6 to extract level-set contours. The two adjacency definitions were tested (images (b and c)), and several thresholds and filtering were used to extract the digital contours in (d,e). Figure 7 presents complementary results to measure the robustness to noise. These results were obtained from the console command `pgm2freeman` (from directory `FrechetAndConnectedCompDemo/demoIPOL_ExtrConnectedReg`). Note that the ordered set of surfels was transformed into a set of pointels by calling method `extractAllPointContours4C`.



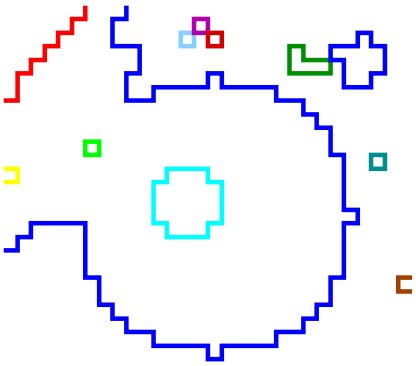
The obtained contours are represented with a freeman chaincode: it is a word whose letters are codes defining the direction when going from a point to another (0 is right, 1 up, 2 left, 3 down). Such a contour is illustrated on the previous floating figure with the starting point A and with the chain code 101212323300. All these experiments can be reproduced from the demonstration source by using the command lines that follows (from the `build` directory).

• **Extraction of contours with different adjacency definitions.** (results given on Figure 6 (a-c) with specific colors to highlight each contour.)

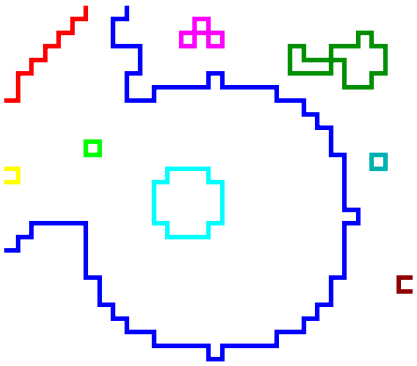
```
./demoIPOL_ExtrConnectedReg/pgm2freeman -image ../demoIPOL_ExtrConnectedReg↩
   /Images/shapeTest.pgm -minThreshold 200  -maxThreshold 255 -badj 0 >↩
```



(a) source image.



(b) interior adjacency.



(c) exterior adjacency.

(d) source image.

(e) contour extraction from gray scale range.

(f) extraction from gray scale range and contour selection.

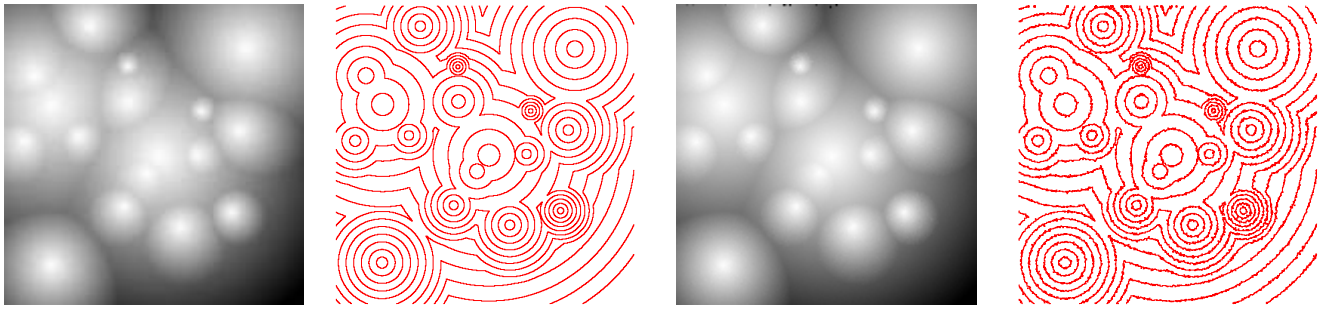
Figure 6: Visualization of the level set contours extracted with the proposed algorithm (given from `pgm2freeman` tool). Images (a-c) illustrates the setting of the interior or exterior adjacency parameter. Image (e) shows the extraction of contours from the Lena image with a threshold step equal to 20 and with minimal contour size equal to 20. Image (f) shows a filtering of such contours from a reference point and a minimal distance (respectively equal to (150,150) and 50).

```
    shapeTestInt.fc
./demoIPOL_ExtrConnectedReg/pgm2freeman -image ../demoIPOL_ExtrConnectedReg↩
    /Images/shapeTest.pgm -minThreshold 200  -maxThreshold 255 -badj 1 >↩
    shapeTestExt.fc
```

The set of contours can be displayed with the source image in background:

```
./demoIPOL_ExtrConnectedReg/displayContours -fc shapeTestExt.fc  -outputFIG↩
    shapeTestExt.fig -backgroundImageXFIG ../demoIPOL_ExtrConnectedReg/↩
    Images/shapeTest.jpg 30 30;
./demoIPOL_ExtrConnectedReg/displayContours -fc shapeTestInt.fc  -outputFIG↩
    shapeTestInt.fig -backgroundImageXFIG ../demoIPOL_ExtrConnectedReg/↩
    Images/shapeTest.jpg 30 30;
fig2dev -L pdf shapeTestExt.fig shapeTestExt.pdf
fig2dev -L pdf shapeTestInt.fig shapeTestInt.pdf
```

• **Extraction of contours given by a threshold range.** Extraction of all digital contours from 0 to 256 by step of 20 gray levels with a minimal size equal to 20 (result given in Figure 6 (d,e)):



(a) source image. (b) level set contours. (c) noisy version of (a). (d) level set contours.

Figure 7: Level set contours extraction on gradient image with (c,d) and without noise (a,b).

```
./demoIPOL_ExtrConnectedReg/pgm2freeman -image ../demoIPOL_ExtrConnectedReg↩
   /Images/lena.pgm -thresholdRange 0 20 128 -min_size 20   > ↩
   lenaContourSet.fc
```

The set of contours can be displayed with the source image in background:

```
./demoIPOL_ExtrConnectedReg/displayContours -fc lenaContourSet.fc  -↩
   outputFIG lenaContourSet.fig -backgroundImageXFIG ../↩
   demoIPOL_ExtrConnectedReg/Images/lenaBG.gif 256 256
```

• **Selection of a set contours from a reference point.** (result given in Figure 6 (d,f))

```
 ./demoIPOL_ExtrConnectedReg/pgm2freeman -image ../↩
    demoIPOL_ExtrConnectedReg/Images/lena.pgm -thresholdRange 0 20 128 -↩
    min_size 20  --contourSelect 150 150 50 > lenaContourSetSelected.fc
 ./demoIPOL_ExtrConnectedReg/displayContours -fc lenaContourSetSelected.fc ↩
     -outputFIG lenaContourSetSelect.fig -backgroundImageXFIG ../↩
    demoIPOL_ExtrConnectedReg/Images/lenaBG.gif 256 256
fig2dev -L pdf lenaContourSetSelect.fig lenaContourSetSelect.pdf
```

The experiments of Figure 7 are obtained with the same command lines (with images `circularGradient.png` and `circularGradientNoise.png`).

3.2 Surface Tracking on Digital 3D Objects

We also illustrate the surface extraction for digital 3D objects. First, a simple set of 3D voxels was generated on Figure 8 to display the connected components obtained with different values of adjacency. Each color represents one connected component. As expected, the exterior adjacency connects components which were disconnected with the interior adjacency. Figure 9 shows other surface extraction (with interior adjacency) from a set of voxels obtained by thresholding image (a).

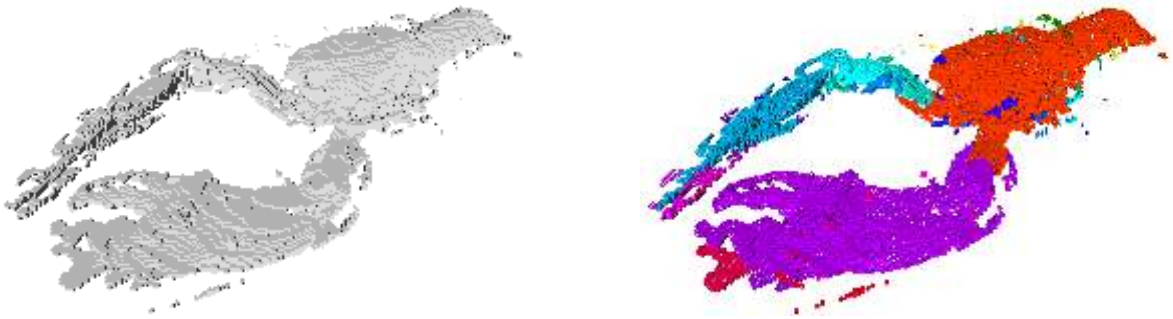
These experiments can be obtained from the demonstration source with the following command that extracts a set of surfels with interior adjacency and export it in a 3D mesh representation (`.off` format):

```
./demoIPOL_ExtrConnectedReg/extract3D -image ../demoIPOL_ExtrConnectedReg/↩
   Images/sample3D1.vol  -badj 0 -output surfelCAdjInt.off -exportSRC ↩
   surfelCSRC.off
```

The threshold can also be set to particular values as for the lobster 3D image of Figure 9:

(a) voxel set (b) extraction with interior adjacency (c) extraction with exterior adjacency

Figure 8: Illustration of surface tracking on a digital 3D object. Each color represents a connected component. For image (b) (resp. (c)) the connected components are obtained with interior (resp. exterior) adjacency.



(a) set of voxels. (b) connected set of connected surfels.

Figure 9: Illustration of connected surfel extraction (b) from a set of voxel (a). Each color represents a connected component defined according to the interior adjacency.

```
./demoIPOL_ExtrConnectedReg/extract3D -image ../demoIPOL_ExtrConnectedReg/↩
    Images/lobster.vol  -badj 0 -threshold 120 255 -output lobIntAdj.off -↩
    exportSRC lobSRC.off
```

4 Minimal Code to Include the Contour Extraction in a C++ Program

To extract the contour directly in your own C++ program you have just to use the following code:

- Include the following header files :

```
// To use image and import:
#include "DGtal/images/ImageContainerBySTLVector.h"
#include "DGtal/io/readers/PNMReader.h"

//To extract connected components
#include "DGtal/topology/helpers/Surfaces.h"
```

- Import an image:

```
typedef DGtal::ImageContainerBySTLVector< Z2i::Domain, unsigned ↩
    char> Image;
Image image = PNMReader<Image>::importPGM( "circleR10.pgm" );
```

- Define the threshold with Binarizer object:

```
typedef IntervalThresholder<Image::Value> Binarizer;
Binarizer b(0, 128);
PointFunctorPredicate<Image,Binarizer> predicate(image, b);
```

- Extract all contours:

```
Z2i::KSpace  ks; //Khalimsky space in 2D
ks.init( image.domain().lowerBound(), image.domain().upperBound↩
    (), true );
SurfelAdjacency<2> sAdj( true );  // Interior adjacency in 2D
std::vector< std::vector<  Z2i::Point > > contours;
//extraction of all the contours
Surfaces<Z2i::KSpace>::extractAllPointContours4C( contours, ks, ↩
    predicate, sAdj );
```

The extraction in 3D images can also be processed in a similar way:

- Include the supplementary header files for the 3D images:

```
// To import volume images:
#include "DGtal/io/readers/VolReader.h"

// To export the display
#include "DGtal/io/Display3D.h"
```

- Import a 3D volume image and define a binarizer:

```
typedef ImageSelector < Domain, int>::Type Image;
Image image = VolReader<Image>::importVol("sample3D1.vol");

typedef IntervalThresholder<Image::Value> Binarizer;
Binarizer b(1, 255);
PointFunctorPredicate<Image,Binarizer> predicate(image, b);
```

- Extract all the sets of connected surfels:

```
// We just have to update the dimension to 3:
Z3i::KSpace ks;
ks.init(image.domain().lowerBound(), image.domain().upperBound(), ↩
    true);

SurfelAdjacency<3> sAdj(  badj );
vector<vector<SCell> > vectConnectedSCell;

// Extract of the sets of connected surfel set:
Surfaces<KSpace>::extractAllConnectedSCell(vectConnectedSCell, ks,↩
     sAdj, predicate, false);
```

Image Credits

All images created by the authors except:

 VolVis distribution of SUNY Stony Brook, NY, USA.[7]

 Standard test image.

References

[1] G. T. Herman, *Discrete multidimensional Jordan surfaces*, CVGIP: Graphical Models and Image Processing, 54 (1992), pp. 507–515. http://dx.doi.org/10.1016/1049-9652(92)90070-E.

[2] T. Y. Kong and A. Rosenfeld, *Digital topology: Introduction and survey*, Computer Vision, Graphics, and Image Processing, 48 (1989), pp. 357–393. http://dx.doi.org/10.1016/0734-189X(89)90147-3.

[3] V. A. Kovalevsky, *Finite topology as applied to image analysis*, Computer Vision, Graphics, and Image Processing, 46 (1989), pp. 141–161. http://dx.doi.org/10.1016/0734-189X(89)90165-5.

[4] J.-O. Lachaud, *Coding cells of digital spaces: a framework to write generic digital topology algorithms*, in Proceedings of 9th International Workshop on Combinatorial Image Analysis (IWCIA'2003), Palermo, Italy, vol. 12 of ENDM, Elsevier, 2003, pp. 337–348. http://dx.doi.org/10.1016/S1571-0653(04)00497-4.

[7] http://labs.cs.sunysb.edu/labs/vislab/volvis-gallery

[5] A. Rosenfeld, *Connectivity in digital pictures*, Journal of the ACM, 17 (1970), pp. 146–160. http://dx.doi.org/10.1145/321556.321570.

[6] ——, *Adjacency in digital pictures*, Information and Control, 26 (1974), pp. 24–33. http://dx.doi.org/10.1016/s0019-9958(74)90696-2.

[7] J. K. Udupa, *Multidimensional digital boundaries*, CVGIP: Graphical Models and Image Processing, 56 (1994), pp. 311–323. http://dx.doi.org/10.1006/cgip.1994.1028.