Union-Find Volume Segmentation

Christophe Fiorio, Jens Gustedt, Thomas Lange

Union-Find volume segmentation

Christophe Fiorio*
LIRMM, 161 rue Ada
F-34392 Montpellier Cedex 5, France
fiorio@lirmm.fr

Jens Gustedt
LORIA, BP 239, campus scientifique
F-54506 Vandoeuvre ls Nancy, France
gustedt@loria.fr

Thomas Lange†
Konrad-Zuse-Zentrum, Takustrasse 7,
D-14195 Berlin, Germany
thomas.lange@charite.de

Abstract

We present an efficient method to segment large 3d images that takes the whole 3-dimensional structure into account. We use a Union-Find data structure to record and maintain the necessary informations during the segmentation process. This structure leads to a volume labelling and facilitates interactive process by linking directly voxels to the volume they belong to. Moreover the global integration of all the data enhances the accuracy of statistical merging criteria.

1 Introduction and Overview

In this paper we present a method to perform real 3-dimensional segmentation and in particular of tomographic images (for a more general issue on medical imaging, see e.g. [Her83, Aya93]). The data in these images are a set of cross-sectional (slice) digital images of a part of a human body. We aim to segment the whole image in volumes such that they correspond to significant part of the image. In this paper we focus on a combinatorial structure and algorithms to do efficiently the volume segmentation, and we are not concerned by the choice of pertinent merging criteria.

Several methods which perform such extraction exist, see [UH91], but the common and non-satisfactory approach is to segment each slice separately and to construct volumes by matching borders of regions of a slice with borders of regions of the precedent and following slices [Boi84, BY98]. A variant is to perform an edge detection on each slice and identify in each of them the edges which match the object that is to be extracted, see e.g. [GL93]. There is another approach described in [MBA93] which take a binary image but where a 3D-edge segmentation is assumed to be done (as for example the one described in [MDMC90]) and perform a topological reconstruction.

Our approach is slightly different, since we will take the 3-dimensional information into account and identify volumes in the image as a whole. So afterwards, no additional

*This work was partially done while Christophe Fiorio and Jens Gustedt were working at the Technische Universitt Berlin

†Supported by the Konrad-Zuse-Zentrum

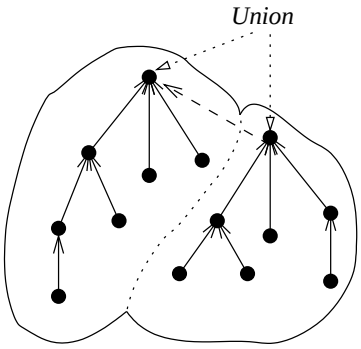

Figure 1: Union operation

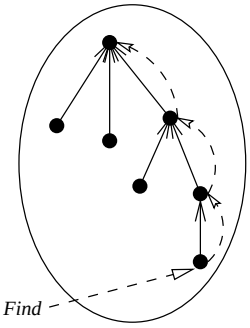

Figure 2: Find operation

processing has to be done, all volume are determined. Moreover we posses some additional statistical informations which was maintained during the process in order to compute merging criteria (such as for example size of the volume). In a 3D imaging software we can easily designate and visualize each volume by selecting an individual voxel.

We define a volume in the classical way as a set of connected voxels, where the connectivity relation is the 6-connectivity relation (see [KR89] for a definition), but the method can easily be extended to the 18 or 26 connectivity. Our method performs *volume growing*, by extending the region growing of the 2D case. We are starting with small volumes and we merge them together according to criteria. These criteria will use statistical informations of the volume to take their decision. Note that our method allows to use local criteria as well as global ones. Indeed our structure provide us with a connected component labeling, so for a given pair of voxels we have local informations related to the voxels themselves, and global informations related to the segment they belong to.

First we shortly present the basics of Union-Find and how it can be applied to image segmentation. Then, our approach is presented and we show how we deal with the entire 3-dimensional image. It follows a description of an application to computer tomographie and a presentation of results. Finally some perspectives are given.

2 Basics of Union-Find

The general Union-Find problem, or more precisely *the disjoint set-union problem*, can be formulated as follows. Given is a set S, the ground-set, of elements that form one-element subsets at the beginning; the goal is to perform arbitrary sequences of Union and Find operations in the best time complexity possible. Here a Union works on two disjoint subsets merging them into one, a Find identifies the subset a certain element belongs to. For an introduction and overview to Union-Find see e.g. [Meh84, GI91]; for recent results see [vKO93].

Efficient implementations of the Union-Find problem use tree data structures to represent sets (i.e. segments in our case), the root of the tree being the representative of the region. To perform an Union operation it is sufficient to link the two roots of the corresponding tree (see Figure 1), creating thereby a new tree. The Find operation identifies the region by finding the root of the tree in an iterative pointer search (see

Figure 2).

In the general case, the best complexity known has been first obtained by Tarjan, see [Tar75], who has shown that general Union-Find algorithms perform in $O(\alpha(n,m)m)$ where α is a very slowly growing function. and $n < m$ are the amounts of calls to a a Union and Find operation respectively. In [Gus98] it is shown that the Union-Find problem can been solve in linear time on a RAM for special classes of graphs, in particular for d-dimensional grids for fixed d and 8-neighborhood graphs of a 2 dimensional grid.

So theoretically there is no problem since an image can be considered as a 2-dimensional grid and a 3-dimensional image as a 3-dimensional grid. But the algorithms described in [Gus98] are very complex and the application to image segmentation is not trivial. But there exist linear algorithms (see [DST92, FG96]) that perform in linear time in the special case of 2-dimensional image segmentation. The theoretical complexity of these algorithms translate very well in short running-time. The method described here is based on the algorithms given in [FG96] (see Algorithms 1 on the following page and 2 on the next page).

Let us now present these algorithms and discuss about their generalization to the 3-dimensional case.

3 Application to the segmentation problem

Clearly the disjoint set-union problem translates very well into segmentation by merging: sets are regions of the image, *find* allows to find the region a pixel belongs to, *union* merges two regions. It is just necessary to take the connectivity constraint of the regions into account. To ensure that the regions stay connected among the different union operations, we always consider two regions of two adjacent pixels. Indeed another important point of Union-Find structure is that algorithms using such a structure are driven by elements, sets are found thanks to the *find* operation. So Union-Find segmentation will be driven by the pixels. Thus we can use both local (i.e. relative to the pixels) and global (i.e. relative to the regions) merging criteria, and this leads to a contour-region cooperation. Since we can identify the region each pixel belongs to in parallel to the segmentation process, such a method provides us with a connected component labeling as well.

At the beginning of the process of region growing, all the pixels form regions of only one element. A scanning strategy (line-by-line – see Algorithm 1 –, linear quadtree – see Algorithm 2 –, random) has to be chosen. It must examine all pairs of pixels of the image, and for each pair decides, thanks to a predicate $\mathcal{P}$, whether or not to perform the merging of the two regions the pixels belong to. Note that the scanning strategy plays an important role in the process and its outcome. A deterministic strategy will lead to better efficiency, but will influence the shape of the obtained regions in particular line-by-line strategy. An ideal strategy would be a random scan ensuring an homogeneous growing of the regions.

This method can – in theory – easily be generalized to 3-dimensional images. The same technique applies, just replace pixel by voxel in the above text. But to implement such an approach, there is a problem due to the limitation of memory. A Union-Find data structure is a greedily space consuming. At the beginning, each voxel is a segment, so one should have the necessary structure for a segment for each voxel. And in addition to the grey level of the pixel and the pointer to the father in the tree, this structure must record all the data needed for the merging criteria, such as e.g. number

Algorithm 1: `ScanLine(`I`)`

Input: an image I with r rows and c columns

for $j = 0$ ***to*** $r-1$ **do**
  for $i = 0$ ***to*** $c-1$ **do**
    if $i > 0$ **then**
      left = `Find(`$I[i-1, j]$`);`
      current = `Find(`$I[i, j]$`);`
      if $\mathcal{P}$(*left,current*) **then** `Union(`*left,current*`);`
    end
    if $i > 0$ **then**
      up = `Find(`$I[i, j-1]$`);`
      current = `Find(`$I[i, j]$`);`
      if $\mathcal{P}$(*up,current*) **then** `Union(`*up,current*`);`
    end
  end
end

Algorithm 2: `mergeSquare(`S,k`)`

Input: an integer k and a square S

if $k = 0$ **then return** ;
$h^- = 2^{k-1}$; $h^+ = 2^{k-1}$;
for *D=NW* ***to*** *SE* **do** `mergeSquare(`S_D,$k-1$`);`
for $i = 0$ ***to*** 2^k **do**
  left = `Find(`$S[i, h^-]$`);`
  right = `Find(`$S[i, h^+]$`);`
  if $\mathcal{P}$(*left,right*) **then** `Union(`*left,right*`);`
end
for $i = 0$ ***to*** 2^k **do**
  up = `Find(`$S[h^-, i]$`);`
  down = `Find(`$S[h^+, i]$`);`
  if $\mathcal{P}$(*up,down*) **then** `Union(`*up,down*`);`
end

of pixels in the region, sum of the grey levels of all the pixels of the region, etc. For example, in our application the size of such a structure is about 32 bytes. So it is difficult to handle all the 3-dimensional image in memory at once. In our application, the size of the image is $512 \times 512 \times 178$, so all the image structure will require about 1.4 Gb of memory!

So, it is necessary to adopt a particular strategy in order to reduce the needed amount of memory space. Moreover such strategy can improve the practical efficiency of Union-Find algorithms (see [FG97]). Let's see now how we use Union-Find structure for the segmentation of 3-dimensional images.

4 Union-Find volume segmentation

As we have already mentioned, we cannot proceed as for the 2-dimensional case and set at the beginning a volume for each voxel. Moreover this would increase the drawback

of a deterministic scanning strategy and decrease the accuracy of statistical merging criteria since a single voxel could be compared to a volume of hundreds of voxels.

In order to decrease the amount of data (i.e. the number of initial volumes) and increase the significance of merging criteria we perform a *weak segmentation* on each slice which reduces the number of initial segments to be integrated in the global Union-Find structure. It will then be possible to feed the entire structure obtained into memory. The general procedure can be summarize as in Algorithm 3. Note that the scanning strategy used to illustrate the process is not the only one possible but it is the simplest. For a discussion on the scanning order, see Section 6.

Algorithm 3: Volume Segmentation

```
   Input: an image I with s slices, r rows and c columns
   for k = 0 to s − 1 do
1      load slice S_k;
2      weak segmentation of S_k;
3      flatten(S_k);
4      integration of weak segments into the global data structure;
   end
5  for z = 0 to s − 1 do
       for j = 0 to r − 1 do
           for i = 0 to c − 1 do
               if i > 0 then
                   left = Find(I[i − 1, j]);
                   current = Find(I[i, j]);
                   if P(left,current) then Union(left,current);
               end
               if j > 0 then
                   behind = Find(I[i, j − 1]);
                   current = Find(I[i, j]);
                   if P(behind,current) then Union(behind,current);
               end
               if k > 0 then
                   up = Find(I[i, j, k − 1]);
                   current = Find(I[i, j]);
                   if P(up,current) then Union(up,current);
               end
           end
       end
   end
6  save the result slice by slice;
```

The steps 1,2, 3 and 4 form the first part of our algorithm. Its goal is to reduce the amount of data in order to be able to deal with the entire 3-dimensional image. Step 4 is done in order to ensure a linear complexity and then a good efficiency. Step 4 implements the global Union-Find Structure which describes the volumes partition.

The second part consist in the step 5 and 6. Step 5 realizes volume segmentation by managing the global Union-Find structure and creating volumes by the merge of the segments of the slices. Step 6 save the result slice by slice.

We will now discuss about the *weak segmentation* and the *flatten* (step 2 and 3). Then we present in details the integration of *weak segments* into the global data structure (step 4).

4.1 Weak Segmentation of Individual Slices

The goal of this step is to reduce the amount of data. To do that, we use the segmentation process in each slice as a preprocessing tool performing a filtering on the original image. But we choose a very restrictive merging criterion such that the regions obtained are still relatively small and no abusive merging are done. This is what we call a *"weak segmentation"*.

The intention for this is that we want to do a real 3-dimensional segmentation in order to obtain a volume description of the image. By doing a complete segmentation of each slice, we would not take the 3-dimensional information into account. In fact we would fall back into classical methods which segment each slice separately and performs a 3d-reconstruction of the image afterwards.

In keeping regions small we ensure that:

- no abusive merging will be performed on any slice,
- the 3-dimensional information is still pertinent.
- 3d criterion will be more significant.

A discussion about the merging criterion used for the weak-segmentation is out of the scope of this paper. In Section 5.2 we give the criterion we have used for the weak-segmentation and which is able to guarantee the properties we want.

For the segmentation process, we use the ones described in Section 3. It is fast and provides a connected component labeling (see [FG96]). This last point is important since this will facilitate the integration of the weak segments into the global data structure. Now, after the weak segmentation, the number of regions in a slice is substantially reduced (see Section 5.3).

4.2 Flattening of the slices

Before integrate these data into the global structure, we will perform an additional process which will ensure a better complexity in the final algorithm and thus better performances. This process, called *flatten the slice* consists in linking all the voxels directly to the segment they belong to[1]. This realizes the component labeling of each individual slice effectively.

Note that the complexity of such a process is linear in the number of voxel of the slice. Indeed, for each voxel we do an iterative pointer search of the root, and on the path followed we link all the voxel directly to the root (*path compression*, see Figure 3). So each edge of the tree, which is not linked to the root or a leaf, will be visited only one time, since after an iterative pointer search all other voxels which are linked to a voxel of this path will find directly the root from it. Each extremal edge (i.e. edge linked to a leaf or the root) will be used one time for each leaf. So the total number of pointer jumps is at most equal to two times the number of edges in the tree which

[1] We recall that a segment (a set) is described by a rooted tree and that the root of a tree is the representative element of the set.

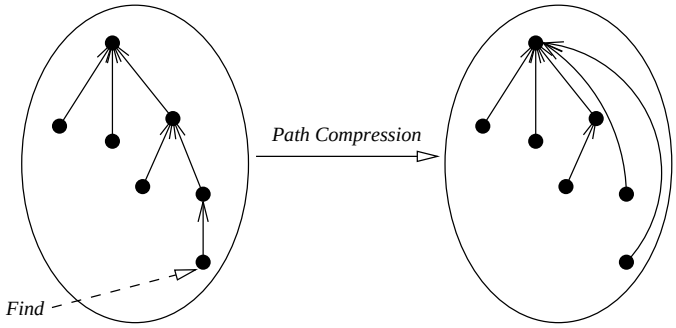

Figure 3: Path compression

is equal to the number of element of the tree minus one, that's give us a linear time complexity. For a complete discussion see [FG96].

Note that the structure of a slice is still big since you always have a structure for each voxel. But we will not keep such a structure for each slice of the 3-dimensional image as it will be shown in the following.

4.3 Global Integration of all the Data

After the weak segmentation (step 2 of Algorithm 3 on page 5) of a slice, we have still a structure for each voxel of the slice. But we no more need each voxel, we have just to incorporate segments determined by the weak segmentation. So our data structure for the entire 3-dimensional will not be as for the 2d case a matrix of Union-Find structures, but a matrix of pointers to Union-Find data structures. Indeed, several voxels, now, belong to the same segment, so can share the same data structure. The integration of the segment into the global data structure consists only to set the pointers of the 3d matrix representing the image to the segment provided by the weak-segmentation of each slice. Algorithm 4 shows how we proceed. Step 1 of Algorithm 4 numbers the regions obtained by the weak segmentation of the slice. Function `CreateVolume`, used step 2 in the algorithm creates a new volume from the region given in parameter and returns a reference on this new volume. This function initializes the volume with the statistical data of the region, as for example the number of voxels of the volume. At last, step 3 set the values of the elements of the matrix.

At the end of the algorithm we get an Union-Find structure which describes the volumes of the image. We can now proceed with step 5 of Algorithm 3 which will perform the volume segmentation.

5 Application to Computer Tomographies

In this section, we present some results obtained by our algorithm on a medical images. One is an image of an heart, the other is a computer tomography of a part of a human body.

Algorithm 4: Integration of the weak segments into the global data structure

```
   Input: A r × c × s matrix I of pointer to Union-Find data structures
          s^th slice S of the image, weak segmented
   id = 0;
1  for j = 0 to r − 1 do
       for i = 0 to c − 1 do
           if Find S[i, j] is the root of the tree then
               set region number to id;
               id = id + 1;
           end
       end
   end
   Initialize an array SR of id pointers on volume data structures;
   for j = 0 to r − 1 do
       for i = 0 to c − 1 do
           if SR[I[i, j] → number] is not initialized then
2              V =CreateVolume(Find(I[i, j]));
               SR[I[i, j] → number] = V;
           end
3          I[i, j, s] = SR[I[i, j] → number];
       end
   end
```

5.1 Description of the Data

For the human body, we have 178 slices of a computer tomography starting at the 8^{th} chest vertebra and going down to the knee. Each slice is a 256 grey levels image of size of 512×512. Figure 4 on the next page shows the 28^{th} slice. It represents a cut through the body at about the 11^{th} chest vertebra. In this 2D image one can easily identifies the vertebra, the kidneys at the right and left of it and the liver.

The real distance between two slices is not the same as the one between two pixels of a slice (about 2 times). So our voxels are not cubic but parallelepipedic.

We thus have a $512 \times 512 \times 178$ image, that is 46.661.632 voxels.

The heart image is a $64 \times 64 \times 64$ image. The real distance in x, y and z is the same.

5.2 The Merging Criteria

In our application we ensure that the properties needed to have a good performance by choosing a global merging criterion that gives a tight bound on the variance of the regions obtained. This criterion computes the quantity $t = \frac{\overline{r_1} - \overline{r_2}}{\sigma} \sqrt{\frac{n_1 n_2}{n_1 + n_2}}$ where $\overline{r}$ means the mean of grey levels of the region r, σ is the standard deviation of the region r_1 and r_2. This quantity follows a Student law and according to a table and a risk parameter (usually about 5%) we have to check if t is less than the value given by the table. The predicate for the weak segmentation will combine this test on t and a check on the size of the new region (which does not be too big).

Besides that, we use two other simple local criteria. They are two threshold values which help us to distinguish the background and a great part of the bones. A more pertinent local information would be a 3-dimensional edge, i.e. the presence of local

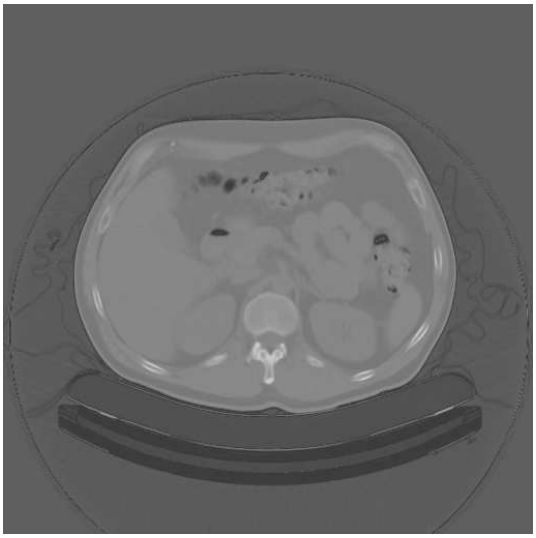

Figure 4: 28^{th} slice of the image

discontinuities as given by such an algorithm described in [MDMC90]. We intend to include such more sophisticated local criteria in the near future to obtain segmentations that respect boundaries of organs even better.

For the volume segmentation, we use less restrictive criterion, but much more sophisticated as the one described in [FN98]. Generally this sort of statistical criterion need a minimum amount of data to be accurate. This is the case since we have done the weak segmentation.

5.3 Statistics (Data Size, Processing Time)

The following table summarizes some statistics. The numbers given are mean values or approximations.

	Heart image	Body tomography
Image size	$64 \times 64 \times 64$	$512 \times 512 \times 178$
mean size of a weak segment	5	10
Memory for one slice	128 Kb	8 Mb
Memory for the 3d matrix	256 Kb	177 Mb
Memory for the 3d UF structure	1.5 Mb	160 Mb
Total time for steps 1,2,3,4	0.85*s*	190*s*
Time for step 5	0.86*s*	170*s*

In this table we can see that the entire structure need less than 20 times the memory used by one one slice. Note that, quite common for medical image, the background takes about one half space of the image (see for example slice of body tomography in Figure 4). So this increases the reduction of memory needed for the 3d structure. Nevertheless the simple fact that weak segments are about 10 pixels is sufficient to achieve a reasonable memory space. Note that this experiments have been done on a Pentium 300 with 512Mb of RAM.

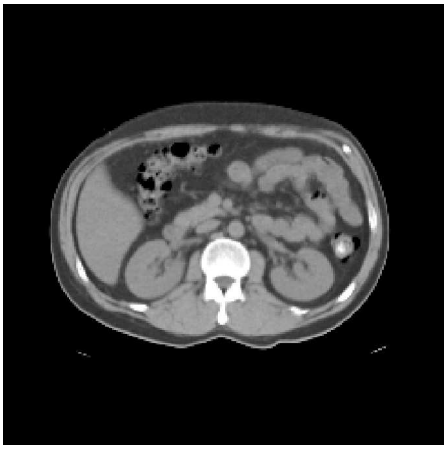

Figure 5: Original slice

Figure 6: Vertical cut

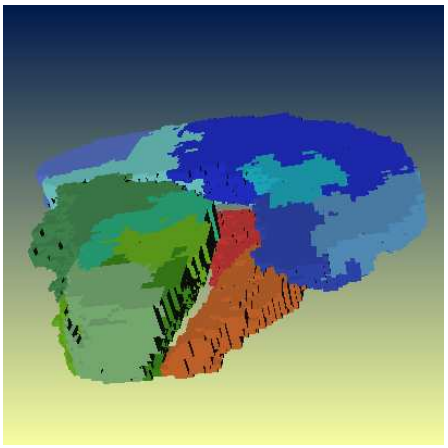

Figure 7: Optical Lobus Union-Find segmented

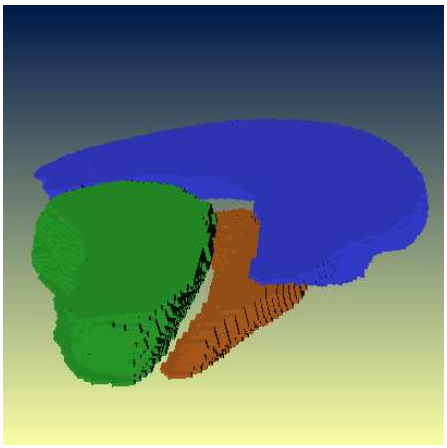

Figure 8: Optical Lobus manually segmented

5.4 Some Results

The following images and projections have been obtained thanks to Amira software[2]. This software includes a volume segmentation module which implements our algorithms [Lan99]. Figure 5 shows one of the 178 slices given as input, Figure 6 shows a vertical cut reconstructed from the data.

In Figure 7 we can see the segmentation of an optical lobus given by our algorithm. The image is made of 21 slices of 128×128 voxels. Compared to the manual segmentation in Figure 8 the result is quite good since, only a few volumes have still to be merged and only a little part is missing.

In Figure 9 we can see the contour of volumes drawn (in green) on a slice. Figure 10 show some of these volumes selected and displayed in a 3d projection. Each volume is

[2] http://amira.zip.de

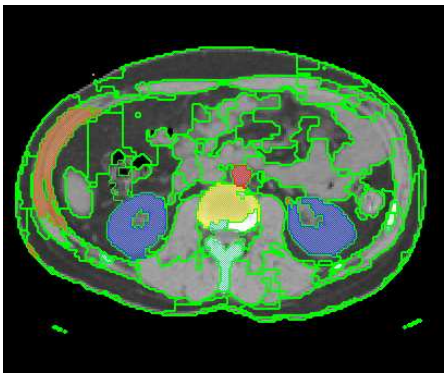

Figure 9: Slice with contours

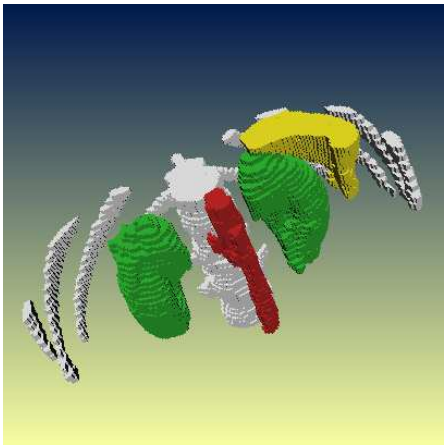

Figure 10: Skeleton, kidneys and aorta

identified by a different colour.

At last, Figure 11 shows the interface of the Amira software. The left window is the display window, upper right window shows the objects pool, middle right window allow to set parameters of the segmentation and lower right window is the Amira shell.

6 Conclusion and Perspectives

In this paper we have presented a new approach to do 3D segmentation. The algorithm shows good results on computer tomography images the processing times are particularly fast. This approach uses a Union-Find data structure in order to deal with the data as a whole. Moreover it provides us with a connected component labeling and we are able to use local and global criteria.

No reconstruction are necessary afterwards, since we have already identified volumes and are able to say for each voxel the volume it belongs to. So to display an organ, for example, it is sufficient to point to a voxel in the image which belongs to it, in order to display the volume representing this organ.

Some improvements are in study, as for example different scanning orders. The slice-by-slice scanning used is the simplest but has a drawback: the volumes to be merged could be disproportionate, especially at the end of the process. We think that a binary-tree order would be better. Moreover this scanning order can lead to a parallelisation of the process.

The way to do the merging can also be improved. As we have already mentioned, the local criteria we used are quite simple and it will be easy to add more sophisticated criteria based on local discontinuities, such the one described in [MDMC90]. Note that the computation time will increase since it will be necessary to pre-compute the map of the 3D-edges.

Another perspective is the use of a Volume Adjacency Graph. This graph can be constructed during the volume segmentation and maintained in parallel to the Union-Find structure. Then after the volume segmentation as described in this paper, we would be able to merge volumes or set of volumes according to the graph. This sort of approach has been already used with success in the 2-dimensional case. So it would be

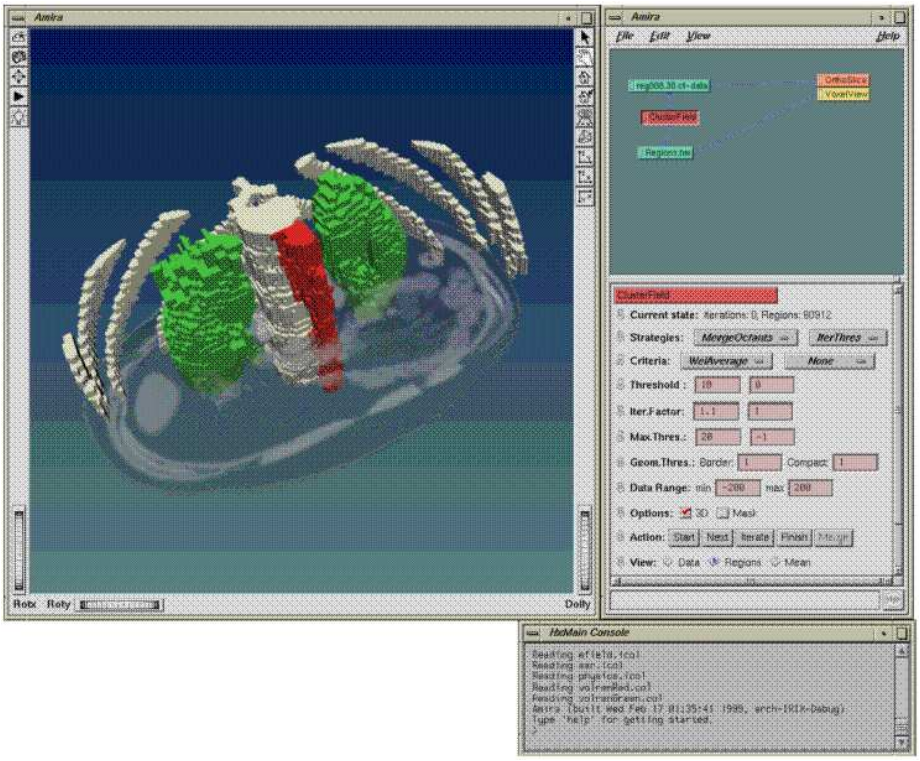

Figure 11: Amira graphic interface

not very difficult to implement it in the 3-dimensional case. The only problem could be the memory usage. But since we have already reduced considerably the memory consuming it could be done without problem.

Aknowledgement

We are grateful to people of the Deutsche Herzzentrum Berlin supplying us with the computer tomographie that forms the base of the pratical results presented in this paper.

References

[Aya93] N. Ayache. Computer vision applied to 3d medical images, results, trends and future challenges. *Int. J. of Computer Vision*, pages 1–20, 1993. référence imcomplète et erronée.

[Boi84] J.-D. Boissonnat. Shape reconstruction from planar cross-sections. *CVGIP*, 44:1–29, 1984.

[BY98] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambride University Press, UK, 1998.

[DST92] Michael B. Dillencourt, Hanan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representation. *J. of the Association for Computing Machinery*, 39(2):253–280, April 1992. Corr. p. 985–986.

[FG96] Christophe Fiorio and Jens Gustedt. Two linear time Union-Find strategies for image processing. *Theoretical Computer Science*, 154:165–181, 1996.

[FG97] Christophe Fiorio and Jens Gustedt. Memory management for union-find algorithms. In Rüdiger Reischuk et al., editor, *14th Symposium on Theoretical Aspects of Computer Science (STACS '97)*, volume 1200 of *Lecture Notes in Computer Science*, pages 67–79, Luebeck, Germany, February 27, March 1 1997. Springer-Verlag.

[FN98] Christophe Fiorio and Richard Nock. Image segmentation using a generic, fast and non-parametric approach. In 10^{th} *International Conference on Tools with Artificial Intelligence*, pages 450–458, Taipe, Taiwan, R.O.C., November 1998. IEEE Computer Society.

[GI91] Zvi Galil and Giuseppe F. Italiano. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys*, 23(3):319–344, September 1991.

[GL93] Muhittin Gökmen and Ching-Chung Li. Edge detection and surface reconstruction using refined regularization. *IEEE trans. Pattern Analysis and Machine Intelligence*, 15(5):492–499, may 1993.

[Gus98] Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theoretical Computer Science*, 203(1):123–141, 1998.

[Her83] G.T. Herman. Special issue on computerized tomography. *IEEE Proc.*, 71:291–435, 1983.

[KR89] T.Yung Kong and Azriel Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing.*, 48:357–393, 1989.

[Lan99] Thomas Lange. Regionenbasierte Segmentierungsverfahren für dreidimesionale biomedizinische Bilddaten. Diplomarbeit, Technische Universität Berlin, März 1999.

[MBA93] Grégoire Malandain, Gilles Bertrand, and Nicolas Ayache. Topological segmentation of discrete surfaces. *Int. J. of Computer Vision*, 10(2):182–197, 1993.

[MDMC90] O. Monga, R. Deriche, G. Malandain, and J.P. Cocquerez. Recursive filtering and edge closing: two primary tools for 3d edge detection. In *Proc. 1st European Conference on Computer Vision*, volume 427, Nice, France, April 1990. Lecture Notes in Computer Science, Springer Verlag: New York.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Springer, 1984.

[Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. of the Association for Computing Machinery*, 22:215–225, 1975.

[UH91] J.K. Udupa and G.T. Herman. *3D Imaging in Medicine*. CRC Press, Boca Raton, FL, 1991.

[vKO93] Marc J. van Kreveld and Mark H. Overmars. Union-copy structures and dynamic segment trees. *J. of the Association for Computing Machinery*, 40(3):635–652, July 1993.