A new split and merge algorithm based on Discrete Map.

Luc Brun and Jean Phillipe Domenger

{ domenger,brun } @labri.u-bordeaux.fr

LaBRI, Université Bordeaux I

351, Cours de la libération 33405 Talence cedex, France.

Abstract

Usually, the segmentation algorithms implementing the split and merge operations are restricted to a split stage followed by a merge stage. In this paper, we present a new split and merge algorithm combining alternatively split and merge operations at each recursive step. This algorithm is based on a data structure called *discrete map* [BD96a]. This data structure allows us to design segmentation algorithm which are almost impossible to implement with usual data structure.

Keywords

Segmentation, split and merge algorithms, discrete map.

1 Introduction

According to Horowitz [HP76] a segmentation of an image X for a uniformity predicate P is a partition of X into disjoint nonempty subsets $X_1, X_2, \ldots, X_n$ such that :

1. $X = \bigsqcup_{i=1}^{n} X_i$

2. X_i is connected for all i in $\{ 1, \ldots, n\}$

3. $\forall i \in \{1, \ldots, n\} \quad P(X_i) = true$

4. $\forall i \neq j P(X_i \cup X_j) = false$ where X_i and X_j are connected.

The conditions (1) and (2) insure that the image is partitioned into a set of regions. The condition (3) insure that each region is homogeneous according to the homogeneity criterion P. The condition (4) insure that all regions are maximal, thus that every merge of two regions will produce a non-homogeneous region. If we split a non-homogeneous region we obtain more homogeneous subregions. Conversely the merge of two regions produce a less homogeneous region than the two merged regions. The region-based segmentation methods can be classified into three main approach : *The Top-down* methods begin with an under-partition of the image and split all regions which do not respect condition (3). The *bottom-up* methods start with an over partition of the image and merge all regions which do not respect condition (4). The *Mixed-method* alternate the split of regions which do not respect condition (3) and the merge of regions which do not respect condition (4) until all regions respect both conditions (3) and (4).

The top-down methods use hierarchical representations of the image as pyramids [Bro82, PRW82] or quadtrees [DRH80, Sam80]. These approaches are based on a recursive decomposition of a square image domain into square sub-domains. The split of a region can be efficiently performed by a refinement of the subdivision while the computation of the adjacency-regions on a tree structure involves complex and costly processing. In order to preserve the tree structure, the mixed approach is restricted to the merging of adjacency regions having a same father, this kind of merger is called *a restricted merge* [HP76].

The usual data structure to implement the merge algorithm is an *array of labels* [Nic95] combined with a *region adjacency graph* (R.A.G) [CMVM86, BF70]. An array of labels associates to each pixel a label such that all the pixels of a given region have a same label. The vertices of the adjacency graph represent the regions of the image and there exists one edge between two regions if they are adjacent. The merge of two regions consists in contracting the edges which link them and suppressing the induced multiply edges. The update of a RAG induced by the split of a region into sub-regions involves many computations. Indeed, the split region must be suppressed of the RAG and all the new sub-regions must be traversed in order to insert their corresponding node and their adjacency edges.

Due to the incompatibility between the structures used by the split and merge algorithm, the mixed approach consists in a first part to alternate split and restricted merge. In a last part, a RAG is generated in order to allow *unrestricted merge* of any adjacent regions (also called *grouping* [HP76]) . An important drawback of this approach is that the incoherent regions created during the split may remain in the partitioning because their existence is only reconsidered after a sequence of split operations.

In this paper, we present a split and merge algorithm where the split and the merge operations may be iterated to any step of the algorithm. The model of *discrete map [BD96a]* used by this algorithm allows us to combine the split and grouping operations without over cost. Moreover, the regions considered in this work are *4-connected* set of pixels having any size or form.

We present in section 2, the data structure encoding the regions of the image. The section 3 presents the homogeneity criterion used by our split and merge algorithm. The section 4 is devoted to the description of the split and merge algorithm.

2 Discrete maps encoding image regions.

An efficient framework to implement split and merge algorithms is provided by the model of *discrete maps* [BD96a]. Discrete map is a mixed model combining a discrete description of the geometry of region boundaries with a Euclidean description of the topology of the image. The advantages of this model are: a correct discrete boundary definition of regions having free geometry and its ability to express the usual split and merge operations in terms of graph modifications. The region splitting can be summed up to the insertion of new edge and the region merging to the suppression of existing edges.

Implicit description of regions is given using an *inter-pixel boundary* representation [AAC95, Fio95]. The set of boundaries is encoded in an alternative discrete plane, the *boundary plane*. Its elements are the points of the Euclidean plane with coordinates of the form:

$$\left(\frac{2k+1}{2}, \frac{2k'+1}{2}\right), \text{ where } k \text{ and } k' \text{ are integers.}$$

Intuitively, the points of the boundary plane correspond to the pixels corners. A point $p = (x_p, y_p)$ of the *image plane* and a point $p' = (x'_p, y'_p)$ of the *boundary plane* are called *half-neighboring points* if $|x_p - x'_p| = |y_p - y'_p| = \frac{1}{2}$. Each point of the image plane has four half-neighboring points in the boundary plane and conversely. The set of *boundary points* is made of boundary points having two half-neighboring points belonging to two different regions of the image. The boundary of an image region r is the subset of boundary points having at least one half-neighboring point in the region r.

As direct result of the Khalimsky topology [KKM90, KKM91], the following properties have been established by Braquelaire and all. [BD96a]:

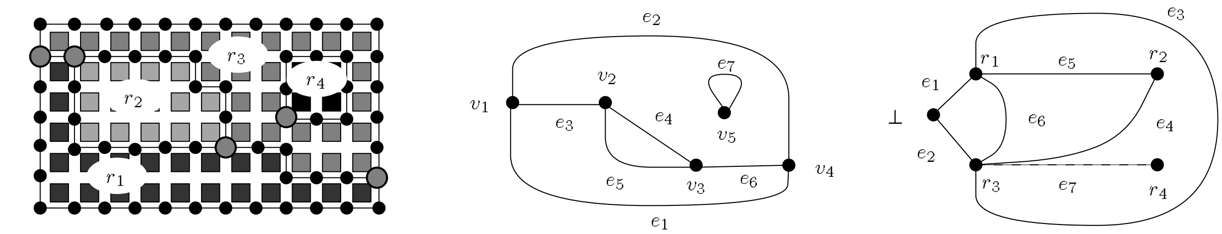
1. the boundary of a region is made of closed Jordan curves defined in the boundary plane, each of these curves validates a discrete Jordan theorem;

2. the boundaries of two 4-adjacent regions share a set of Jordan arcs having at least two boundary points.

The first property allows the reconstruction of the region using scan-line or seed-fill algorithms [BD96a]. Therefore an homogeneity criterion can be evaluated on a region.

The both previous properties implies that if the image is subdivided into simply 4-connected regions then the set of region boundaries forms a *discrete map* which is the drawing of a *topological map* [Tut63] in the Khalimsky topological plane. The drawing of a *face* of a discrete map is an image region. The drawing of a *vertex*, called a *node*, is a boundary point belonging at least to the boundaries of three different regions. The drawing of an *edge* called a *segment*, is a 4-connected path of boundary points shared by two different regions. One discrete map defines only image partition into simply connected regions. In order to represent any image (eg. image with holed region) an inclusion relation between a face and its included discrete map must be considered. Hence, the topology of an image can be represented by a set of discrete maps augmented with an inclusion relation.

The topological maps provides a more powerful framework than a RAG. Indeed, if we consider the dual of topological maps completed by the inclusion relation, each edge of the dual graph connects two adjacent regions. Moreover, each of these edges identifies one segment. Note, that the contraction of the multiple edges of the dual topological map produces the RAG. In figure 1, an image and its dual topological map is displayed. For instance, the merge of the region r_3 and r_4 is done by suppressing the edge e_7 and the vertex r_4.



(a) Discrete maps in the boundary plane (b) Topological maps (c) Dual topological map

Figure 1: The part (a) shows the image and its region-boundary set. The big circles are the nodes and the black circles are segment points. The part (b) shows the topological maps, there exists two graph because r_3 is holed by r_4. The part (c) displays the dual topological maps of the image, The exterior of the image is represented by the region $\perp$. The edge e_7 drawn by a dashed line corresponds to the link induced by the inclusion relation.

Now, we present briefly an estimation of the memory requirements of discrete maps (a complete estimation is given in [BD96a]). The geometry of region boundaries can be efficiently encoded by an array $\mathcal{B}$ of four bits, if the geometrical size of the image is $H \times W$ then the size of this array is $(H+1) \times (W+1)/2$ bytes. This array encodes for each boundary points the links which connect it with its neighboring boundary points. A boundary points of a segment is linked with exactly two other boundary points. Using $\mathcal{B}$, a segment can be efficiently traversed starting from one of its extremity nodes. Topological maps can be efficiently encoded using **combinatorial map**[Cor75, GHPV89]. The memory requirements of the dual topological map, expressed in integer size, is $E + 4 * V$ where E is the number of edges and V the number of vertex of the dual topological maps.

Let r be an image region, we denote by $\mathcal{V}(r)$ the set of adjacent regions of r, by $\partial(r)$ the boundary of the region r and by $d^o(r)$ the degree of the vertex associated with r in the dual topological map. Let s be a segment of the discrete map, we denote by $|s|$, the number of points belonging to s.
In a *merge problematic*, the required operations and their time complexity are:

1. **Computation of** $\mathcal{V}(r)$,*the time complexity is linear in function of* $d^o(r)$.

2. Selection of $r' \in \mathcal{V}(r)$ according to the evaluation of an homogeneity criterion $hc(r, r')$.

3. **Computation of** $\partial(r) \cap \partial(r')$, *the time complexity is mean linear in* $\frac{d^o(r)+d^o(r')}{2}$.

4. **Suppression of $\partial(r) \cap \partial(r')$, (see the algorithms defined in [BD96b]).** *Since, the geometrical suppression of $\partial(r) \cap \partial(r')$ is done by removing its points in the array $\mathcal{B}$ and since the topological suppression is direct, the time complexity of this operation is linear in $|\partial(r) \cap \partial(r')|$.*

5. **Updating of the inclusion relation**, *the time complexity is* 1 *because they consist in linked two linked list of infinite faces.*

The most expensive operation is the fourth operation, thus the complexity of the merge problematic is in $\mathcal{O}(|\partial(r) \cap \partial(r')|)$ if we expect the evaluation of $hc(r, r')$.
In a *split problematic*, the required operations and their time complexity are:

1. **Traverse r in order to produce the set of splitting segment** $S = \{s_1, \ldots, s_n\}$, *the time complexity is* $|r|$

2. **For each** $s_i \in S$.

   (a) **Creation of two nodes respectively located to the extremities of s_i.** *The creation of a node in a point p requires the identification of the segment containing p. This operation is done in a time proportional to the mean length of a segment of the discrete map.*

   (b) **Effective insertion of** s_i,*same time complexity than segment suppression.*

3. **Updating of the inclusion relation**, *the time complexity is lower than $d * f$ where f is the number of discrete maps previously included in r, and d is the diameter r.*

Since the sum of time complexity of stage 3 and 4 is lower than $|r|$, then the complexity of the split problematic is in $\mathcal{O}(|r|)$.

3 Parameters attached to regions

The model of discrete map is useful for segmentation algorithms only if this model can provide efficiently information about regions and segments. This information has to allow segmentation algorithms to decide which segments have to be removed or where segments need to be added. Thus, the model should provide efficiently information which allows to decide if a region is too or not enough homogeneous. We measure the homogeneity of a region r with the squared error defined by:

$$\boldsymbol{SE}(C_r) = \sum_{x \in C_r} g(x) \|x - \mu(r)\|^2 \tag{1}$$

where C_r denotes the feature space associated to region r, this feature space may be a set of values between $\{0, \ldots, 255\}$ for greyscale images or a three dimensional set for color images. The value $g(x)$ is the number of pixels with color x in region r and $\mu(r)$ is the mean of the set C_r. In the following we will denote $\boldsymbol{SE}(r)$ for $\boldsymbol{SE}(C_r)$.

The squared error of a region computes the sum of squared distance of each color to the mean color of a region. In other words the squared error represents the error produced by the approximation of C_r by its mean $\mu(r)$. Thus a low squared error corresponds to an homogeneous region while a high squared error corresponds to a non-homogeneous region. The definition of the squared error given by equation 1 does not allow an efficient update of the squared error of regions during merge operations. A better approach consists in defining the squared error from moments of order 0, 1 and 2 of a region.

Definition 1 *Let r be a region, the moments or order 0, 1 and 2 of r respectively denoted by $M_0(r)$, $M_1(r)$ and $M_2(r)$ are defined by the following equations:*

$$\begin{array}{rcl} M_0(r) & : & \textit{number of pixels contained in region} R_f \\ M_1(r) & = & \sum_{(v_1,\ldots,v_n) \in C_f} (v_1, \ldots, v_n) \\ M_2(r) & = & \sum_{(v_1,\ldots,v_n) \in C_f} (v_1^2, \ldots, v_n^2) \end{array} \tag{2}$$

Where n denotes the dimension of the feature space (usually one or three) and $(v_1, \ldots, v_n)$ denotes coordinates in the feature space.

If we merge two regions r_1 and r_2 to create the region $r_1 \cup r_2$ the relation between the moments of r_1 and r_2 and the moments of $r_1 \cup r_2$ is defined by :

$$\forall\ i \in \{0, 1, 2\} \quad M_i(r_1 \cup r_2) = M_i(r_1) + M_i(r_2) \tag{3}$$

Using the moments M_0,M_1 and M_2, the squared error, the variances and the mean of a region r may be computed by the following equations:

$$\begin{array}{rcl} \mu(r) & = & \frac{M_1(r)}{M_0(r)} \\ var_i(r) & = & \frac{M_2(r)_i}{M_0(r)} - \mu_i^2(r) \\ \boldsymbol{SE}(r) & = & M_0(r) \sum_{i=1}^{n} var_i(r) \end{array} \tag{4}$$

where $var_i(r)$ denotes the variance of the feature space C_r along the axis i. In the same way, $M_2(r)_i$ denotes the i^{th} coordinate of the two-order moment of r.

In our implementation we have attached to each region its moments of order 0, 1 and 2. We choose to compute by need the attributes of a region r and to mark them as invalid when r is split. Due to the linearity of moments defined by equation (3), moment updating does not require to traverse merged regions.These attributes can thus be efficiently updated along merge operations. Moreover, using equation 4 the squared error, the mean and the variances of a region can be deduced immediately from moments.

In order to allow users to design other efficient merge-score, we also attach and compute by need the length and the mean gradient of each segment. Such parameters allow the design of merge-score based on the common boundaries of two regions.

4 The split and merge algorithm

Most of split and merge algorithms use hierarchical data structures [CP79, PRW82, CMVM86] such as quadtree [DRH80, Sam80] or pyramids [Bro82, PRW82]. These data structures involve two main limitations to the split and merge algorithms :

- The split of a region is performed by dividing a square defining it into four sub-squares. This process creates a "square aspect" in the final segmentation which is difficult to break (see [BCR90] for further details)
- Since the merge algorithm should not break the hierarchical data structure, the merge has to be restricted so as to respect the tree-structure.

Obviously, the restricted merge is less powerful than the unrestricted merge. This lack of efficiency may create the same artifacts than the one created by a split algorithm followed by a grouping step. For example the region formed by two half circles on Figure 2, has been created by the first split algorithm. Let us suppose that this region is not meaningful and can't be suppressed with a restricted merge. If we split further the initial domain, this region may be considered as enough homogeneous, and will not be split further. Thus, the expected contour splitting this region will not be created and the grouping algorithm following the split sequence will be unable to find the correct

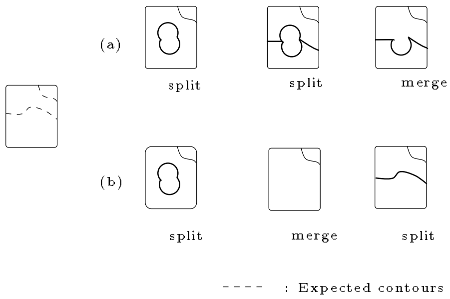


Figure 2: *Line (a) a sequence of splits and restricted merge followed by a grouping. Line (b) a sequence of split and unrestricted merge.*

partition. On the contrary, if we perform a merge operation after the first splitting step, the wrong region will be removed and the split process following this merge step will be able to find the expected contours. This unwanted behavior of split and merge algorithms using a restricted merge is illustrated by a real world image in Figure 5.

The data structure defined in section 2 allows us to overcome the limitations imposed by a hierarchical data structure. First, our data structure allows us to encode free-geometry regions. This property induce a better behavior of our split algorithm which will split further more homogeneous regions. Moreover, the free geometry of our regions allows us to avoid computation burden to break the initial partition into squares. Moreover our data structure allows us to merge any region r with a region belonging to its neighborhood $V(r)$. This last point allows us to remove the distinction between the restricted and the unrestricted merge operations.

We saw in last section that the squared error provides an efficient tool to measure the homogeneity of a region. We have used this homogeneity criterion to design a new split algorithm. This algorithm described in [Bru96] splits the cluster C_r corresponding to the region r into a set of clusters $C_1, \dots, C_p$. Each cluster C_i will produce a set of regions $r_{i_1}, \dots, r_{i_q}$ such that all pixels of regions $r_{i_1}, \dots, r_{i_q}$ have feature coordinates which belong to cluster C_i. The homogeneity of regions produced by the split algorithm are controlled by the following requirements on clusters C_i :

$$\sum_{i=1}^{p} \boldsymbol{SE}(C_i) < \alpha \boldsymbol{SE}(C) \tag{5}$$

$$\forall i,j \in \{1,\dots,p\} \quad \boldsymbol{SE}(C_i \cup C_j) > \beta \boldsymbol{SE}(C) \tag{6}$$

The condition (5) ensures that all regions are homogeneous enough with respect to the initial region. The condition (6) ensures that the split algorithm will not produce two adjacent regions with too close features. The symbols α and β denote two thresholds which may be parameterized by users.

Once the set C_f is split into a set of clusters $C_1, \dots, C_p$ we have an implicit partition of the region. As a matter of fact, two adjacent pixels belong to different regions if and only if their features do not belong to a same cluster. A functionality of our implementation inserts edges in the data structure according to this implicit partition. Thus segmentation algorithms only have to provide a function which decides if two pixels belong to a same region or not.

Parameters used by segmentation algorithms to decide if a region has to be split further or not may also be used to decide if two given regions has to be merged. We have designed a merge algorithm (see figure 3) using the squared error as homogeneity criteria. This algorithm scans a list of region L and merges the adjacent regions until all merges will produce a region with a squared error greater than a given threshold. Note that the algorithm may merge two regions with one of them which do not belong to L. The region resulting from the merge will be added to L. This property allows us to correct regions created by previous iterations of the split algorithm.

```
merge(list of regions L)
{
    do
    {
        (r, r') = ArgMin_(r1∈L, r2∈V(r1)) SE(r1 ∪ r2)
        if(SE(r ∪ r') > ε)
            return
        r ∪ r' = merge_two_regions(r, r')
        remove r and r' from L
        insert r ∪ r' in L
    }
    while(ever)
}
```

Figure 3: *Our merge algorithm*

The split and merge algorithm displayed in figure 4 alternate split steps and merge steps. We have slightly restricted the merge algorithm to the regions created by the split and merge algorithm. This restriction allows us to modify only regions which form a partition of the domain to be segmented. Thus we avoid modifications of regions created by previous segmentation algorithms, these regions being supposed correct.

```
split_merge(region r_ini)
{
    list of regions L',L = ∅
    region r_max = r_ini
    do
    {
        split(r_max)
        L' ← regions created by the split algorithm
        merge(L')
        update L and add L' to L
        r_max = ArgMax_{r∈L} SE(r)
    }
    while(SE(r_max)/SE(r_ini) > ε)
}
```

Figure 4: *Our split and merge algorithm*

5 Conclusion

The split and merge method has first been introduced by Horowitz and Pavlidis [HP76]. This method was implemented with a quadtree data structure which involves many limitations to the method. We have presented a set of algorithms built on a new data structure. This data structure allows us to overcome all limitations imposed by quadtrees. The algorithms based on our model have a greater degree of freedom, and are thus more powerfull. The split and merge algorithm presented in section 3 is an illustration of the new abilities of segmentation algorithms based on the structure presented in section 2. Our segmentation algorithms are enough efficient to allowed interactive segmentation. This new ability allow users to guide the segmentation process and thus to obtain the desired partition. Finally, the flexibility of our data structure allows searchers to implement quickly new segmentation algorithm based for example on other criteria than the squared error.

References

[AAC95] E. Ahronovitz, J.P Aubert, and Fiorio C. The star-topology: a topology for image analysis. In *5th DGCI Proceedings*, pages 107–116, 1995.

[BCR90] M. Bister, J. Cornelis, and A. Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognit Letter.*, 11(9):605–617, Sept. 1990.

[BD96a] J.P. Braquelaire and J.P. Domenger. Representation of region segmented images with discrete maps. Technical report, RR-112797 LaBRI, available on http://www.labri.u-bordeaux.fr/LaBRI/Publications/index.html, June 1996.

[BD96b] L. Brun and J.P. Domenger. Incremental modifications on segmented image defined by discrete maps. Technical report, RR-112696 LaBRI, available on http://www.labri.u-bordeaux.fr/LaBRI/Publications/index.html, may 1996.

[BF70] R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial intelligence*, 1:205–226, 1970.

[Bro82] J.D. Browning. Segmentation of pictures into regions with tile-by-tile method. *Pattern Recognition*, 15(1):1–10, 1982.

[Bru96] L. Brun. Split, merge and grouping : a new approach for color image quantization. Submitted to Computer Vision Journal, may 1996.

[CMVM86] F. Cheevasuvut, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on a split and merge procedure. *CVGIP*, 34:268–281, 1986.

[Cor75] R. Cori. Un code pour les graphes planaires et ses applications. Thèse d'état de l'université Paris VII, and Astŕisque 27, 1973 and 1975.

[CP79] P. Chen and T. Pavlidis. Segmentation by texture using a co-occurence matrix and a split and merge algorithm. *Computer Graphics, and Image Processing*, 10:172–182, 1979.

[DRH80] R.C. Dyer, A Rosenfeld, and S Hanan. Region representation: Boundary codes from quadtrees. *ACM: Graphics and Image Processing*, 23(3):171–179, March 1980.

[Fio95] C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation*. Thèse de doctorat, Université Montpellier II, 24 novembre 1995.

[GHPV89] M. Gangnet, J.C. Hervé, T. Pudet, and J.M. VanThong. Incremental computation of planar maps. In *Proc. of SIGGRAPH'89*, 1989.

[HP76] S.L. Horowitz and T. Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of The Association for Computing Machinery*, 23(2):368–388, April 1976.

[KKM90] E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of applied Math. and Stocastic Analysis*, 3:27–55, 1990.

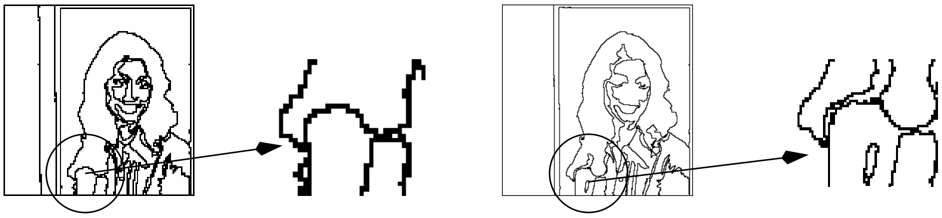
[KKM91] T.Y. Kong, R. Kopperman, and P.R. Meyer. A topological approach to digital topology. *Amer. Math. Monthly*, 98(10):901–917, 1991.

[Nic95] C. J. Nicol. A systolic approach for real time connected component labeling. *Computeer vision and Image understanding*, 61(1):17–31, January 1995.

[PRW82] M. Pietikainen, A. Rosenfeld, and I. Walter. Split and link algorithms for image segmentation. *Pattern Recognition*, 15(4):287–298, 1982.

[Sam80] H. Samet. Region representation: Quadtrees from bondary codes. *ACM : Graphics and Image Processing*, 23(3):163–170, March 1980.

[Tut63] W.T. Tutte. A census of planar maps. Canad.J.Math., 15:249–271, 1963.



(a) (b)

Figure 5: *A restriction of the merge algorithm to the faces generated by the preceding split algorithm has involved the remaining of the two faces on the left harm of the girl (see Part a). Our split and merge algorithm avoids such artifacts (see Part b).*

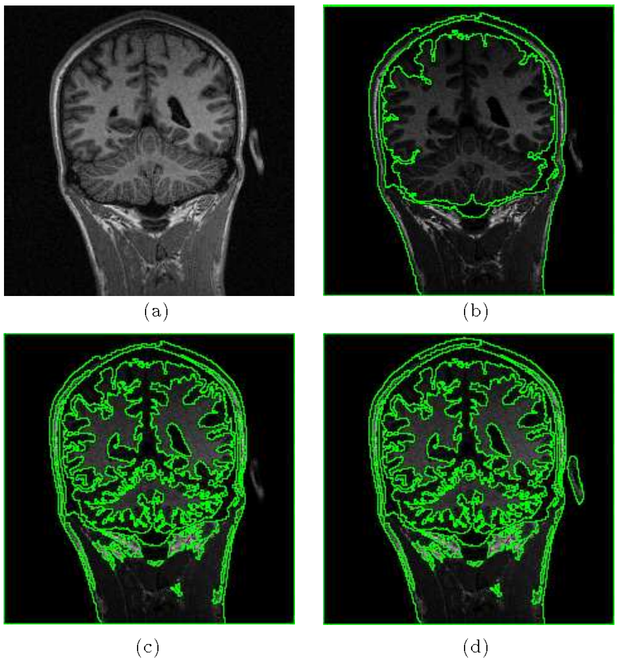


Figure 6: *(a) Original image. (b) to (d) Three iterations of our split and merge algorithm.*