On Integrating Iterative Segmentation by Watershed with Tridimensional Visualization of MRIs

Romaric Audigier, Roberto Lotufo
Fac. of Elec. and Comp. Engin. – UNICAMP
C.P. 6101, 13083-852, Campinas (SP), Brazil
{audigier,lotufo}@dca.fee.unicamp.br

Alexandre Falcão
Inst. of Computing – UNICAMP
C.P. 6176, 13083-970, Campinas (SP), Brazil
afalcao@ic.unicamp.br

Abstract

This work presents two fast and iterative methods that integrate segmentation by watershed and three-dimensional visualization, while the classical approach is to separate these two processes. The user-aided segmentation is based on iterative watershed, efficiently implemented using the Image Foresting Transform (IFT). The first proposed algorithm consists of extracting segmented structures' borders during the segmentation and updating them at each iteration. Structure visualization is achieved by border projection. The second algorithm updates the image directly from the changes of the segmented scene occured in each step. The two methods were implemented in C and tested with various Magnetic Resonance Images. The results are satisfactory as compared to the classical approaches. The incremental image algorithm is a little faster than the classical one while the incremental border algorithm, although a little slower in segmenting, allows a very fast rendering, very suitable for object manipulation. Thus, the proposed methods provide a good feedback to user.

1. Introduction

The use of digital images has increased during the last decades in a lot of fields, especially in medical imaging. In medicine, tomographs (CT, PET, SPECT as well as MRI) provide a non-invasive way of observing the human body's internal structures. In many applications (like surgery, diagnosis or teaching) a three-dimensional reconstruction and a 3D-visualization are required for qualitative analysis. Furthermore, if quantitative analysis is needed, data must be previously segmented. Indeed, image segmentation allows to determine volume, surface or other measures of the structures of interest, which are very useful for users. Although there exists systems using automatic segmentation, in general segmentation is manually done by specialists, due to the lack of robustness and correctness of the automatic process when different kinds of images are analyzed. Unluckily this manual technique is normally very tedious and time-consuming. Moreover, there is no guaranty of repeatability. Thus, the semi-automatic approach is seen to be a reasonable compromise between both previous approaches: It offers repeatability, it is not so time-consuming for the user and is user-guided, therefore accurate. In such interactive processes, 3D reconstruction and visualization help the user validate the segmentation results or correct them. The traditional approach for providing visual feedbacks to user is to separate the interactive system in two independent processing components: segmentation and then visualization. Systems generally do not merge these two processes.

Watershed method is a well-known and powerful technique of Mathematical Morphology for object segmentation. As we will see in section 2, the whole scene data must be processed for this purpose. Visualization also needs to scan this data. Thus, the idea of this work is to integrate both processes for sparing time. First we present the method of watershed (section 2) and two variations of its implementation by Image Foresting Transform (IFT for short). Then, two new methods of integrated visualization are proposed in section 3. Finally, we describe the experiments and results we obtained with MRIs and other test images (see section 4).

2. Iterative Segmentation by watershed

2.1. Watershed via IFT: Overview and notations

The watershed approach, first introduced by Beucher, Lantuéjoul and Meyer [4, 5], is a simple yet powerful tool of Mathematical Morphology for image segmentation. Intuitively, the watershed algorithm can be compared with a flooding simulation. In this context, an input gray-scale image (generally the morphological gradient of the original data) is considered as a topographic surface where holes are punched in the marker regions. These regions can be found automatically or pointed by the user. After associating each marker with a color, imagine that colored waters go through the holes and enter the topography to inundate it at a uniform rate. Dams are built to prevent the merging of different colored waters. Finally, these dams form the watershed lines between the segmented regions also called 'catchment basins'. For more details about the watershed transform, see also these references [8, 15].

The Image Foresting Transform (IFT), a graph-based technique recently introduced by Falcão et al. [12] and based on Dijkstra's algorithm [7], provides a sound framework for the implementation of image processing operators. Among them can be cited the watershed transform of interest in this paper, distance transforms, fuzzy connectedness-based segmentations, multi-scale skeletonization and connected operators [11, 6, 10, 17]. It has been shown that the implementation of the watershed by IFT is very efficient and results in an optimal solution [13] and can easily be used in 3D as well. The optimality is guaranted by using an ordered queue, as we will see.

In the IFT context, an image (or a 3D scene) can be seen as a graph, where each pixel (or voxel) is a node. After defining a neighborhood N (commonly chessboard or city-block neighborhood), arcs are placed between adjacent pixels. With a given image I and a set S of seed pixels, the IFT calculates an optimal partition of the pixels-nodes in trees of minimal cost, also named a spanning forest: trees grow from the seed pixels and cannot contain any cycle. For this purpose, three attributes are assigned to each node s of a tree: a path cost $C[s]$ (cost of the path from the seed to this node), a root $R[s]$ (seed of the tree) and a predecessor $P[s]$ (node which precedes s in the path). After cost, root and predecessor initializations, a loop for emptying the priority queue Q starts. This ordered queue has been filled firstly with the seed pixels. The pixel s of highest priority, i.e. lowest cost, is removed from Q with its definitive attributes (cost, root and predecessor). The potential path cost c of each neighbor t of s (such as $t \in N(s)$) is evaluated considering that the path to t is the concatenation of the path to s and the arc $\langle s, t\rangle$. If the evaluated cost c is lower than the already assigned cost $C[t]$, the three attributes of t are updated. If not already in the queue, the pixel t is inserted in with priority c. Else, only its position in Q is updated. The algorithm finishes when the queue is empty. Thus, each seed grows and forms a tree of pixels according to the minimum cost criterion. Note that a path-cost function f is needed for evaluating the path cost in t denoted by $f(\pi_t)$. In the particular case of the watershed implementation, this function consists in choosing the *maximum* between the gray-level of t and the cost $C[s] = f(\pi_s)$ of s when the evaluated path π_t is the concatenation of the path π_s and the arc $\langle s, t\rangle$. When a path contains only one voxel, it is a 'trivial path' and has cost $f(\langle s\rangle)$. Each seed constitutes a trivial path whose cost is specified by the user. In the case of watershed, it is generally zero. It is important to observe that because of the priority queue, a pixel removed from the queue has always its definitive attributes. Furthermore, when a pixel is in the queue, it is disputed by its neighbors which propose competitive costs.

2.2. Iterative watershed

A new iterative version of the IFT, called DIFT for Differential IFT, has just been proposed by Bergo and Falcão [2, 3, 9]. This algorithm allows to compute sequences of IFTs in a differential way. At each iteration, trees may be marked for deletion and seeds added for a new dispute among the remaining roots. The optimum forest (costs, roots and predecessors) is updated, without running the algorithm from the beginning. Note that the DIFT algorithm does not process the whole image (or scene) again, but only the pixels (or voxels) whose optimum values need to be updated and, therefore, takes time proportional to the number of these pixels. This is particularly interesting in the case of a segmentation by DIFT-watershed. After a first segmentation, new seeds are added to refine the segmentation and/or 'eliminators' which indicate the incorrect parts to be resegmented.

Pseudocode 1: *DIFT*

1. $\forall t \in I, \quad e(t) = 0;$
2. $(C, P, F) \leftarrow TreeRemoval(C, R, P, S, E);$
3. $\forall t \in F, \quad Insert(t, Q, C[t]); \quad e(t) \leftarrow 1;$
4. **While** $IsEmpty(Q) = FALSE,$
5. $\quad s \leftarrow RemoveMin(Q); \; e(s) \leftarrow 2;$
6. $\quad \forall t \in N(s),$ **If** $e(t) \neq 2$**,**
7. $\quad\quad c \leftarrow f(\pi_s \cdot \langle s, t\rangle);$
8. $\quad\quad$ **If** $c < C[t]$ **or** $P[t] = s,$
9. $\quad\quad\quad C[t] \leftarrow c; \; R[t] \leftarrow R[s]; \; P[t] \leftarrow s;$
10. $\quad\quad\quad$ **If** $e(t) = 1, \; Remove(t, Q);$
11. $\quad\quad\quad Insert(t, Q, c); \; e(t) \leftarrow 1;$

TreeRemoval(C,R,P,S,E):

1. $D = \{\}; \; F = \{\}; \; T = \{\};$
2. $\forall t \in E,$
3. $\quad r \leftarrow R[t];$
4. $\quad$ **If** $C[r] \neq \infty,$
5. $\quad\quad D \leftarrow D \cup \{r\}; \; C[r] \leftarrow \infty; \; P[r] \leftarrow nil; \; Insert(r, T);$
6. **While** $IsEmpty(T) = FALSE,$
7. $\quad s \leftarrow Remove(T);$
8. $\quad \forall t \in N(s),$
9. $\quad\quad$ **If** $P[t] = s, \; C[t] \leftarrow \infty; \; P[t] \leftarrow nil; \; Insert(t, T);$
10. $\quad\quad$ **Else**, **If** $R[t] \notin D, \; F \leftarrow F \cup \{t\};$
11. $\forall t \in S,$ **If** $f(\langle t\rangle) < C[t],$
12. $\quad F \leftarrow F \cup \{t\}; C[t] \leftarrow f(\langle t\rangle); P[t] \leftarrow nil; R[t] \leftarrow t;$

The pseudocode of the DIFT (see the above Pseudocode 1) utilizes the previously introduced notation. We describe it briefly. The inputs are the image (I), its previous attribute maps (costs C, roots R and predecessors P), a set of seeds to be added (S) and a set of eliminators (E). In each DIFT iteration, voxels are initialized with a 'not processed' status ($e(t) = 0$, line 1). When a voxel is in the priority queue, it assumes an 'in processing' status ($e(t) = 1$, l. 3 and

11). And when it finally leaves the queue with definitive attributes, it gets a 'processed' status ($e(t) = 2$, $l.$ 5). After status initialization, the trees marked by eliminators of E are cleared in the function $TreeRemoval$. Clearing the tree means initializing its nodes' costs to infinite and predecessors to nil. Initializations begin from the roots of these trees ($l.$ 2-5) and propagate to neighbors by means of a FIFO queue T ($l.$ 6-10). The function returns the updated cost (C) and predecessor (P) maps and the set F of seeds and frontier voxels. These frontier voxels are neighbors of cleared voxels. They will be considered as starting points for propagation in the principal loop of the algorithm. This loop is similar to IFT's one (see section 2.1), except in line 8: the root of voxel s is propagated to neighbor t when the proposed path cost c is *lower* than the previously assigned $C[t]$ *or* when s is the predecessor of t (result from the previous DIFT iteration). This new condition validates the correction of the iterative algorithm (see [9]). In fact, all costs of the tree 'descendants' of a voxel were calculated in function of the cost of this voxel. As this voxel is leaving the queue, it means that its attributes may have changed, then all its descendants' attributes have to be reevaluated.

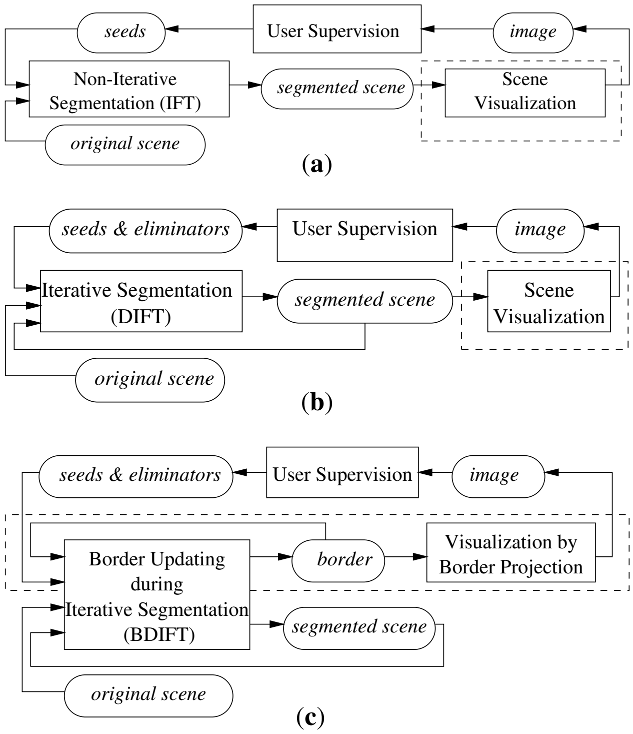

Figure 1. (a) Non-iterative segmentation with scene visualization. (b) Iterative segmentation with scene visualization. (c) Iterative segmentation with border visualization.

3. Interactive 3D-Visualization

As we saw in the introduction, in interactive segmentation systems, 3D visualization is traditionally separated from the segmentation process (see Figure 1(a)). After segmented, the scene is rendered and the user, until he/she is satisfied with the result, can repeat the two processes. If the segmentation is iterative like the DIFT, the segmentation process is accelerated, which increases system's interactivity (see Figure 1(b)). We proposed two ways of integrating visualization with this iterative segmentation. The first method extracts the objects' borders during the DIFT then only updates and renders them at each iteration. The second approach compares the segmentation results of two iterations to update the image ('image' stands for *2D image* or rendition whereas 'scene' stands for *3D data*). Note that in this work, we only considered projective visualization of 3D-scenes and focused on volume rendering (or direct volume rendering) because it processes directly on voxels as DIFT does. Moreover, it does not use any surface modeling. Generally, volume rendering techniques can be divided in two categories: the object-order methods (like voxel splatting) and image-order methods (like raycasting) [18]. Our first algorithm uses only voxel projections (see section 3.1) while the second one combines the two approaches as we show in section 3.2.

3.1. Incremental Borders

This method is based on the idea that in opaque model visualization, only voxels of the objects' border are visible, from any point of view. Therefore, only these voxels need to be projected. It's a type of shell rendering [19] that avoides processing the objects' internal voxels. As in our typical NMR scenes, only 3% of scene voxels belong to borders, border rendering should represent a considerable gain in rendering speed. The novelty and challenge of this technique are the extraction and updating of the objects' borders during the segmentation process. As shown in Figure 1(c), these tasks are encapsulated in the iterative segmentation algorithm (DIFT), now called 'Border DIFT' (or BDIFT for short).

During each iteration of BDIFT, the set of border voxels B must be updated. By definition, a voxel s belongs to an object's border if s belongs to an object and at least one voxel t of its neighborhood $N(s)$ belongs to the background. In other words, if we denote by λ the labeling function which associates non-zero labels with object seeds and the zero-label for background seeds, voxel s belongs to border if $\lambda(R[s]) \neq 0$ and $\lambda(R[t]) = 0$. The 'belong-to-border' property of a voxel depends on its root and its neighbors' ones. Thereby, the BDIFT algorithm is a little more complex than the DIFT one. At each iteration, only the voxels which have their attributes changed may potentially modify the border. These changed voxels are precisely the voxels which enter the queue and then leave it to assume new attributes. So, we analyze now how elements are removed and others added to correctly update the border (see Pseudocode 2).

Initialization and $TreeRemoval$ are the same as in DIFT algorithm. Yet, observe that when inserting seeds and frontier voxels in the queue Q, we systematically remove these elements of the border set B (l. 5). Similarly in the main loop, when a voxel is inserted in the queue, it is removed from border (l. 15). The propagation principles are identical as in DIFT: When a voxel s leaves the queue, if its neighbor t has not yet its definitive value, s tries to propagate its root (l. 9-16). Potential cost c is evaluated and proposed to t. If c is lower than the current cost or the predecessor of t is s, attributes of t are updated (l. 11-15): t has been 'dominated' by a new tree. Else, it is a 'brave resister' to 'invasion' and is inserted in set A if not in the queue (l. 16). On the other hand, when the neighbor t of s has already a definitive value ($e(t) = 2$), it is possible to test if s or t belongs to border (l. 17-19). If the belong-to-border property is satisfied for s, it is inserted in the border, else, the border test is applied to t. After the main loop, a BDIFT-specific loop must be processed (l. 20-22). Border tests are applied to the resister voxels. These resisters are voxels which did not enter the queue because their costs were optimal. Although they did not change attributes, these voxels can affect the border set because they may be neighbors of changed voxels (the dominated). For example, it can occur that a former object-voxel s has been dominated by a new background-tree but its neighbor t has resisted and remains as an object-voxel. It is clear that t has become a border voxel, but nothing in the main loop can manage this exception. Indeed, this resister never entered the queue but could potentially enter it. Until the loop has not finished, there is no way to evaluate t or, by reciprocity, s belong-to-border property, because t is never considered as a definitively valued voxel.

Pseudocode 2: *Border DIFT*

1. $\forall t \in I, \quad e(t) = 0;$
2. $(C, P, F) \leftarrow TreeRemoval(C, R, P, S, E);$
3. $\forall t \in F,$
4. $\quad Insert(t, Q, C[t]); \quad e(t) \leftarrow 1;$
5. $\quad$ **If** $t \in B, \quad B \leftarrow B/\{t\};$
6. **While** $IsEmpty(Q) = FALSE,$
7. $\quad s \leftarrow RemoveMin(Q); \quad e(s) \leftarrow 2;$ **If** $s \in A, \quad A \leftarrow A/\{s\};$
8. $\quad \forall t \in N(s),$
9. $\quad\quad$ **If** $e(t) \neq 2,$
10. $\quad\quad\quad c \leftarrow f(\pi_s \cdot \langle s, t\rangle);$
11. $\quad\quad\quad$ **If** $c < C[t]$ **or** $P[t] = s,$
12. $\quad\quad\quad\quad C[t] \leftarrow c; \quad R[t] \leftarrow R[s]; \quad P[t] \leftarrow s;$
13. $\quad\quad\quad\quad$ **If** $e(t) = 1, \quad Remove(t, Q);$
14. $\quad\quad\quad\quad Insert(t, Q, c); \quad e(t) \leftarrow 1;$
15. $\quad\quad\quad\quad$ **If** $t \in B, \quad B \leftarrow B/\{t\};$
16. $\quad\quad\quad$ **Else, If** $e(t) = 0, \quad A \leftarrow A \cup \{t\};$
17. $\quad\quad$ **Else**,
18. $\quad\quad\quad$ **If** $\lambda(R[s]) \neq 0$ **and** $\lambda(R[t]) = 0, \quad B \leftarrow B \cup \{s\};$
19. $\quad\quad\quad$ **Else, If** $\lambda(R[s]) = 0$ **and** $\lambda(R[t]) \neq 0, \quad B \leftarrow B \cup \{t\};$
20. $\forall s \in A, \quad \forall t \in N(s),$
21. $\quad$ **If** $\lambda(R[s]) \neq 0$ **and** $\lambda(R[t]) = 0, \quad B \leftarrow B \cup \{s\};$
22. $\quad$ **Else, If** $\lambda(R[s]) = 0$ **and** $\lambda(R[t]) \neq 0, \quad B \leftarrow B \cup \{t\};$

3.2. Incremental Image

This second strategy of visualization aims to build an image incrementally, as the DIFT builds a segmented scene. Assuming a fixed point of view between two successive iterations of DIFT, we can state that the differences between the respective images (projective views) of a scene are only due to the changes occured in voxels' labels (and not to scene rotation). The main idea of this method is to analyze changes in segmentation to update the image, as shown in Figure 2.

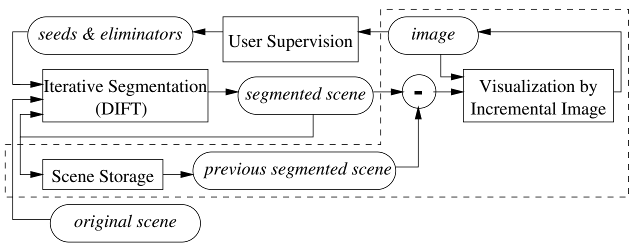

Figure 2. Iterative segmentation with incremental visualization

After a new segmentation iteration, a voxel can either appear or disappear in the image. In the latter case, the voxel s assumes a background label ($L_i[s] = 0$) whereas it previously assumed an object label ($L_{i-1}[s] \neq 0$). In the former case, two situations are possible: s previously assumed a background label ($L_{i-1}[s] = 0$) and now assumes an object label ($L_i[s] \neq 0$), or it simply belonged to another object ($L_{i-1}[s] = \pi \neq 0$ and $L_i[s] = \lambda \neq 0$). Note that to distinguish objects of interest, we generally assigne one color to each object. Consequently, although the voxel of the latter situation remains visible, the image must be updated with respect to the color map.

Pseudocode 3: *Incremental Image*

$\forall s \in I,$
$\quad \lambda \leftarrow L_i[s]; \quad \pi \leftarrow L_{i-1}[s];$
$\quad$ **If** $\lambda \neq \pi,$
$\quad\quad (d, p) \leftarrow VoxelProject(s);$
$\quad\quad$ **If** $\lambda \neq 0, \quad \forall q \in F(p),$
$\quad\quad\quad$ **If** $d \leq D[q], \quad D[q] \leftarrow d; \quad V[q] \leftarrow s;$
$\quad\quad$ **Else**, $\quad \forall q \in F(p),$
$\quad\quad\quad$ **If** $V[q] = s, \quad (\, D[q], V[q]\,) \leftarrow dRaycast(q, L, d);$

To handle these cases, we propose a hybrid algorithm mixing image-order and object-order techniques (see Pseudocode 3). For each voxel s of the scene I, previous and current associated labels are compared ($L_{i-1}[s]$ and $L_i[s]$ respectively). If the voxel assumes a new label (see l. 3), the function $VoxelProject(s)$ computes its projection onto the image plane according the given visualization angles and returns the distance d from the voxel to the image plane as well as the reached pixel's index p (l. 4). In the case of 'appeared' voxel s ($\lambda \neq 0$, l. 5-6), if this voxel is nearer from the image plane than the voxel that was previously projected into pixel p, the image is updated, i.e. depth-buffer D and visible voxel index map V. Observe the wide condition in line 6: It allows updating the image when the voxel only changed its object label but did not really appear. In the case of 'disappeared' voxel s (l. 7-8), we must verify if this voxel s was visible in the image, i.e. if it was projected in pixel p. If so, we must look for the voxel that is visible at this pixel. For this purpose, we cast a ray from the pixel p into the labeled (or segmented) scene L, according to the fixed visualization angles. The ray walks until it encounters an object voxel (with non-zero label). Depth-buffer D and visible voxel index map V are updated accordingly. Note that the raycasting processed by the function $dRaycast(q, L, d)$ is an optimized version of raycasting: The ray is directly cast at a distance d of the image plane and starts its walking from there, as we know that the future visible voxel is necessarily beyond the disappeared voxel. This hybrid method allows to correctly perform three types of changes in any order (see Figure 3). Note that in the pseudocode we utilized a footprint F for splatting the projected voxels and avoiding holes in the image. In this case, we have to check and update all pixels q in the footprint $F(p)$ of p (see l. 5 and 7), not simply pixel p. A more detailed description of the proposed algorithms and results is available in reference [1].

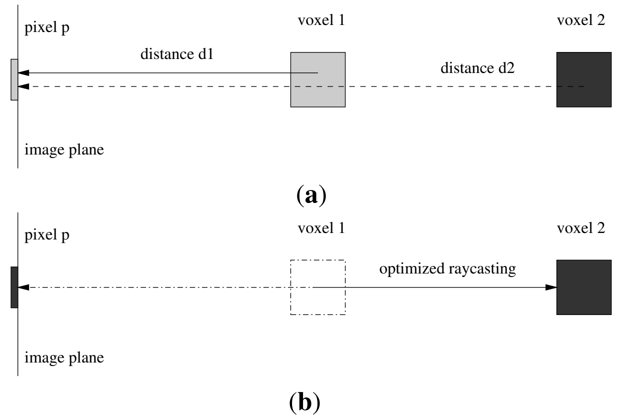

Figure 3. Incremental Image. (a) Voxels 1 and 2 have appeared. We project them in any order. As $d_1 \leq d_2$, $V[p] = 1$ and $D[p] = d_1$. Voxel 1 is visible. (b) Voxel 1 has disappeared. We project it and because $V[p] = 1$ we cast a ray from it until the remaining voxel 2 is found. Finally $V[p] = 2$ and $D[p] = d_2$. Voxel 2 is visible.

4. Experiments and Results

4.1. Experiments

All the described techniques were implemented in C for comparison. We utilized data structures without saving memory space. For example border structure is a doubly linked list which uses the size of two original scenes, although needs actually less than 3% of the size of a scene. It permits a fast access ($O(1)$) to border voxels whenever one voxel is inserted or removed. The ordered queue was implemented with a circular queue using FIFO buckets according to reference [6]. Five experiments were done to compare the performance of five different approaches of interactive segmentation & visualization: (1) IFT with visualization by Scene Projection (SP for short), like in Figure 1(a); (2) DIFT with SP (Figure 1(b)); (3) BDIFT with Border Projection (BP) (Figure 1(c)); (4) DIFT with Incremental Image (II) (Figure 2); and finally (5) DIFT followed by the border extraction (independent process) and BP. This last experiment aims to verify that the encapsulated border extraction in BDIFT is faster than a brute force independent extraction algorithm like in experiment (5). Voxel projections were followed by a typical 3×3 splatting to avoid holes in images. Besides, visualization is performed without perspective (because orthogonal projection is sufficient in most medical applications). Finally, the images were shaded with the Phong's illumination model [16]. All experiments were compiled with Microsoft Visual C++ 6.0 and tested by a 2.4GHz Pentium 4 (1GBytes RAM) on Windows 2000. Processing times are real times. The set of test scenes is shown in Table 1. The size of these 23 scenes varies from 3 to 32 Mvoxels, with a typical size of 7 Mvoxels for common neurology applications. Gray-levels in tested MRIs are encoded in 16 bits, but in 8 bits in other scenes (CT-type or synthetic). As we saw, sets of seeds are necessary at each segmentation iteration. Different seed sets obviously produce different segmentation. Moreover, the same seed set added in a different order can result in different segmentations (see the DIFT pseudocode). Hence, a procedure must be defined for each image (see Table 2) and applied in all 5 tests for comparison.

4.2. Results

Before comparing the different approaches of the five experiments, we analyzed the performances of the algorithms for segmentation on the one hand and for visualiza-

Name	Type	Size	Nb. of voxels
scn1,scn2,scn3	IRM	181x217x181	7 109 137
scn4	IRM	225x175x173	6 811 875
scn5	IRM	234x183x195	8 350 290
brain,brain2,brain3	IRM	181x217x181	7 109 137
brain211	IRM	361x217x181	14 178 997
brain121	IRM	181x433x181	14 185 513
brain112	IRM	181x217x361	14 178 997
brain122	IRM	181x433x361	28 292 653
brain212	IRM	361x217x361	28 279 657
brain221	IRM	361x433x181	28 292 653
knee	RX	171x193x100	3 300 300
geom	synthesized	200x200x100	4 000 000
geom211	synthesized	399x200x100	7 980 000
geom121	synthesized	200x399x100	7 980 000
geom112	synthesized	200x200x199	7 960 000
geom122	synthesized	200x399x199	15 880 200
geom212	synthesized	399x200x199	15 880 200
geom221	synthesized	399x399x100	15 920 100
geom222	synthesized	399x399x199	31 680 999

Table 1. Test scenes

Iteration	1	2	3	4	5	6	7	8
brain	660	730	1030	1230	1400	1550	1955	2105
brain2	1150	1350	1550	1750	1950	2105	–	–
brain3	750	950	1150	1350	1550	1750	1950	2105

Table 2. Examples of procedures for segmenting MR scenes. The table indicates the cumulative number of voxels-seeds automatically inserted at each iteration. Many seeds may be share the same label.

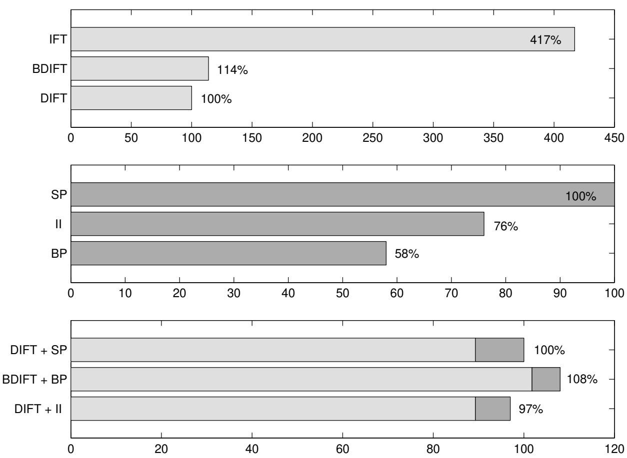

Figure 4. (a) Mean relative time of a segmentation iteration (regardless to the first one) using: IFT, BDIFT or DIFT algorithms. (b) Mean relative time of the visualization process (at each iteration following the first one) using: Scene Projection (SP), Incremental Image (II) or Border Projetion (BP). (c) Mean relative time of a segmentation & visualization iteration (regardless to the first one) for experiments 2, 3 and 4.

tion on the other hand. For a typical MRI like '*brain*', the IFT spends approximately 16 seconds for each iteration. Although DIFT and BDIFT are a little slower (17 s. and 21 s. respectively) at the first iteration, they are much faster (less than 4.5 s.) at the subsequent iterations. After calculating the mean of the relative times of these processes for all test scenes, we confirmed this tendency. As Figure 4(a) shows, IFT algorithm is more than 4 times slower than DIFT, while BDIFT is a little slower (14%) than DIFT. Indeed, BDIFT spends a little more time for extracting the border during the segmentation. But according to the results of experiment 5, we affirm that this extra time is not as long as the time for a brute-force border extraction: BDIFT saves about 20 % of the time for a DIFT followed by a complete border extraction. In fact, BDIFT only updates the border and does not create a border from scratch.

Comparing the three visualization approaches applied in our set of test scenes, we can see in Figure 4(b) that an Incremental Image spends, in mean, 76% of the time for a Scene Projection (typically 300 ms for '*brain*'), whereas a Border Projection spends only 58% of SP time. This result is in accordance with the fact that border contains about only 3% of the scene's voxels. Yet, BP does not save so much time as expected because the voxel access for projection is discontinuous unlike in SP (where accesses are consecutive). Note that II method is particularly efficient when few changes occur in segmentation: it's the case of a few test images. In practice, the user spends more time in refining segmentation details than in resegmenting large areas. Hence this method can be very useful in applications such as the segmentation of the ventricle, the pons-medulla and the cerebellum (see the renditions on Figure 6).

At last, we can analyze the performances of the different strategies of joint segmentation & visualization. Figure 5 shows the total time of the two processes at each iteration when each of the five strategies is applied on '*brain*' (see the renditions on Figure 7). First, we can easily observe that the method with non-iterative segmentation (experiment 1) is much slower (about 16 s) than the others (less than 5s). Second, as we already said, the experiment 5 (brute-force border extraction) is not so efficient (about 2.8s per iteration) as experiment 3 (BDIFT + BP). However, the performances of experiments 2-4 are comparable (aproximately 2 s for each iteration). In fact, for the entire set of scenes (see Figure 4(c)), approach 3 (BDIFT + BP) spends 8% more time than approach 2 (DIFT + SP) whereas approach 4 (DIFT + II) spends 3% less time than approach 2. Indeed, the visualization methods are not preponderant in the total time. It's why BP and II which appeared much more efficient than

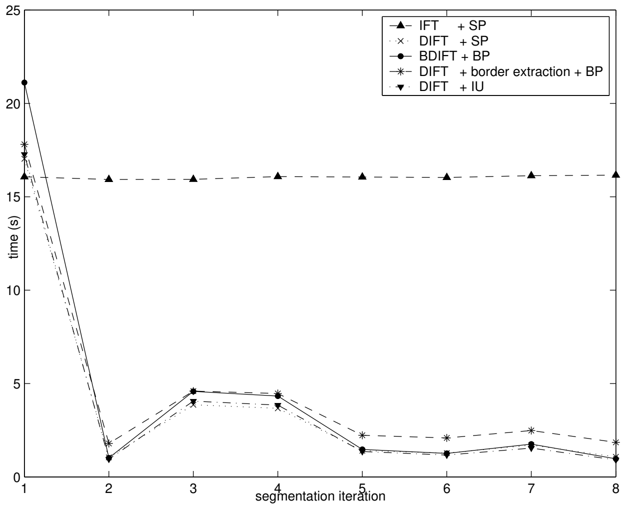

Figure 5. Total time for segmentation & visualization of scene '*brain*' at each iteration of the procedure, according to experiments 1–5.

SP, do not improve (or not significantly) the efficiency of the approaches 3 and 4. Though, the approach 3 has the advantage of storing the border. It allows a rapid manipulation of the object when segmented: After a segmentation iteration, the user can observe the result and rotate the object to decide whether he/she will continue the iterative segmentation. It is particularly significant when the scene is bigger. For example, the scene '*geom222*' takes 1.3s to be rendered, as its border needs only 460 ms. It results that the limit between real time and interactive time (1s according to [18]) can be crossed, depending on the ratio of border voxels and object voxels.

5. Conclusion

As opposed to the traditional approach, we succeeded in integrating both segmentation and visualization of 3D scenes to give to the user a rapid feedback in an interactivity context. Indeed, we proposed two new algorithms for visualization which integrate the DIFT, an iterative segmentation method. Both algorithms are much faster (about 4 times) than the classical approach that uses segmentation by IFT. The first one assumes that the visualization angles are fixed and constructs an image incrementally. It is a little faster than the method that uses the entire scene projection and is a good option for refining a segmentation. The second one extracts the objects's border during their segmentation (BDIFT). It is, therefore, a little slower than a DIFT, because the border is updated during the DIFT. However, it allows a very fast border rendering of the result. It is very useful when the user wants to manipulate the object, and even more when the object is large (and the border not so).

Although the results do not present a significant improvement in relation of non-integrated visualization method, this attempt of integrating both segmentation and visualization is refering to larger questions about interactive segmentation: How to improve interactivity? And how to improve a supervised segmentation? Other indicators than the visual feedback could guide the user in the iterative segmentation process. For example, volume, surface, connected components and use of atlas could help the user choose new seeds and decide whether or not the segmentation is correct. These indicators would be more objective for the user than a sole (subjective) observation of an image. Interestingly, with a converse approach (*analyze* the data to find more easily a transfer function for better *visualization*) Lürig and Ertl [14] proposed a morphology based hierarchical analysis to estimate the optical properties of the volume to be rendered by mean of the transparency visualization model. The present integration of visualization with segmentation shows that one can profit of segmentation process, processing all together data for visualization process and, why not, for determining new indicators, which would help the user in segmenting, giving him/her more feedbacks: Data *analysis* and *visualization* for a better *segmentation* of the objects of interest.

Acknowledgements

I would like to thank the Dept. of Neurology, University of Campinas for the medical images used in the experiments. This work was supported by CAPES.

References

[1] R. Audigier. *Segmentação e Visualização Tridimensional Interativa de Imagens de Ressonância Magnética*. Master thesis, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas (SP), Brazil, Feb. 2004. Supported by CAPES.

[2] F. B. A.X. Falcão. The iterative image foresting transform and its application to user-steered 3d segmentation. *Proc. of SPIE on Medical Imaging 2003: Image Processing*, 5032:1464–1475, Feb. 2003.

[3] F. Bergo. *Segmentação Interativa de Volumes Baseada em Regiões*. Master thesis, Instituto de Computação - Universidade Estadual de Campinas, Campinas (SP), Brazil, Feb. 2004.

[4] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation*, Rennes,France, 1979.

[5] S. Beucher and F. Meyer. The morphological approach to segmentation: the watershed transform. In E. R.

Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 433–481, New York (NY), USA, 1993. Marcel Dekker, Inc.

[6] B. da Cunha. *Projeto de Operadores de Processamento de Imagens usando a Transformada Imagem-Floresta*. Master thesis, Instituto de Computação, Universidade Estadual de Campinas, Campinas (SP), Brazil, 2001.

[7] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] E. Dougherty and R. Lotufo. *Hands-on Morphological Image Processing*. SPIE–The International Society for Optical Engineering, Bellingham (Washington), USA, Aug. 2003.

[9] A. Falcão and F. Bergo. Interactive volume segmentation with differential image foresting transforms. *IEEE Trans. on Medical Imaging*. To appear.

[10] A. Falcão, L. Costa, and B. da Cunha. Multiscale skeletons by image foresting transform and its applications to neuromorphometry. *Pattern Recognition*, 35(7):1569–1580, 2002.

[11] A. Falcão, B. da Cunha, and R. Lotufo. Design of connected operators using the image foresting transform. *SPIE on Medical Imaging*, 4322:468–479, Feb. 2001.

[12] A. Falcão, J. Stolfi, and R. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(1):19–29, Jan. 2004.

[13] R. Lotufo and A. Falcão. The ordered queue and the optimality of the watershed approaches. *5th International Symposium on Mathematical Morphology*, pages 341–350, June 2000.

[14] C. Lürig and T. Ertl. Hierarchical volume analysis and visualization based on morphological operators. In *Proc. of the IEEE Conference on Visualization '98*, pages 335–341, Research Triangle Park (North Carolina), USA, Oct. 1998.

[15] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Processing*, 1(1):21–46, 1990.

[16] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311–317, June 1975.

[17] R. Torres, A. Falcão, and L. Costa. A graph-based approach for multiscale shape analysis. *Pattern Recognition*, 37(6):1163–1174, June 2004. To appear.

[18] J. Udupa and G. Herman. *3D Imaging in Medicine*. CRC Press, Boca Raton (Florida), USA, 1991.

[19] J. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics and Applications*, 13(6):58–67, 1993.

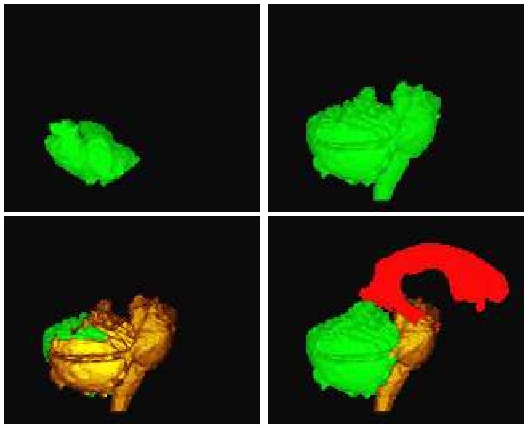

Figure 6. Sagittal view of the segmented 'spare' scene '*brain3*' (4 iterations). Objects of interest are: cerebellum (green), medulla (gold) and lateral ventricles (red).

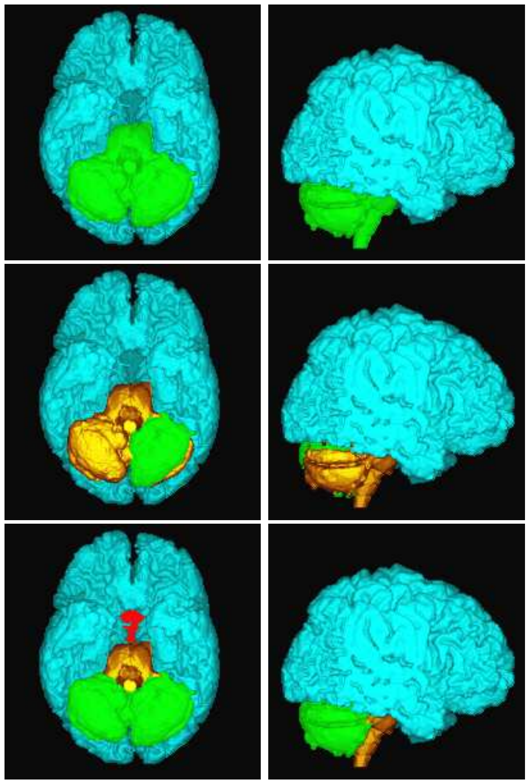

Figure 7. Axial and sagittal views of segmentated scene '*brain*' (3 last iterations). Objects of interest: the same as in Figure 6 and brain (turquoise).