Incremental Watershed Cuts: Interactive Segmentation Algorithm with Parallel Strategy
Quentin Lebon, Josselin Lefèvre, Jean Cousty, Benjamin Perret

To cite this version:
Quentin Lebon, Josselin Lefèvre, Jean Cousty, Benjamin Perret. Incremental Watershed Cuts: Interactive Segmentation Algorithm with Parallel Strategy. 2024. hal-04536919

HAL Id: hal-04536919
https://hal.science/hal-04536919
Preprint submitted on 8 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Incremental Watershed Cuts: Interactive Segmentation Algorithm with Parallel Strategy

Quentin Lebon¹, Josselin Lefèvre², Jean Cousty¹, Benjamin Perret¹

¹LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454, Marne-la-Vallée, France
²Thermo Fisher Scientific, Bordeaux, France

ABSTRACT

In this article, we design an incremental method for computing seeded watershed cuts for interactive image segmentation. We propose an algorithm based on the hierarchical image representation called the binary partition tree to compute a seeded watershed cut. Additionally, we leverage properties of minimum spanning forests to introduce a parallel method for labeling connected component. We show that those algorithms fit perfectly in an interactive segmentation process by handling user interactions, seed addition or removal, in linear time with respect to the number of affected pixels. Run time comparisons with several state-of-the-art interactive and non-interactive watershed methods show that the proposed method can handle user interactions much faster than previous methods with a significant speedup ranging from 15 to 90 on both 2D and 3D images, thus improving the user experience on large images.

© 2024 Elsevier Ltd. All rights reserved.

1. Introduction

In image segmentation, the aim is to divide an image into significant regions. The watershed (WS) is a classical approach to image segmentation where an image is seen as a topological relief which is partitioned into catchment basins associated to the minima of the relief. The WS is an important step of many image analysis pipelines and is still widely used in conjunction with modern machine learning methods such as deep-learning, usually as a pre-processing or a post-processing (see e.g. Machairas et al. (2015); Arbelaez et al. (2011); Eschweiler et al. (2019); Wolf et al. (2017); Lux & Matula (2019)). The WS was first defined on images represented as vertex-weighted graphs Vincent & Soille (1991); Beucher & Meyer (1993). It has then been extended to edge-weighted graphs Cousty et al. (2009) through the notion of WS cut, which is a solution to a global combinatorial optimization problem linked to the Minimum Spanning Tree (MST) problem. WS can also be generalized to hierarchical watersheds, i.e. a hierarchical image representation describing how WS regions progressively merge into the large scale components of the image Najman & Schmitt (1996); Meyer (2012). In particular, the authors of Cousty et al. (2013a); Najman et al. (2013) have shown that WS cuts and WS hierarchies can be computed from a prototypical hierarchical representation called the Binary Partition Tree (BPT).

WS is often considered as a low level-segmentation method because it is highly subject to over-segmentation: as each minimum of the image induces a catchment basin, WS segmentations usually contain a lot of small regions. This problem can be solved by a weakly supervised variation of the WS method called seeded WS. In this case, the minima of the image are replaced by seeds, which can be automatically generated or produced by a user: the number of regions in the result is then equal to the number of seeds.

When seeds are generated by a user, an interactive interface is generally used, allowing the user to modify, add or delete seeds until a satisfactory result is obtained. Traditional seeded WS algorithms don’t handle this incremental process, and the WS segmentation must be completely recomputed after each user interaction, even if it only affects a few pixels. While this may be acceptable for small images, it can seriously degrade the user experience when processing large images and especially 3D images such as RMI or FIB-SEM images. In this context, the authors of Falcão et al. (2004) have proposed the Differential Image Foresting Transform (DIFT) for incrementally updating minimum spanning forests Falcão et al. (2004), which can correspond to WS segmentation, when seeds are modified in the
context of interactive 3D images segmentation.

Interactive segmentation has also been studied in the context of hierarchical representations, such as component trees Passat et al. (2011); Carlinet & Geraud (2015); Ngoc et al. (2023) or segmentation hierarchies Salembier & Garrido (2000). In this case, the hierarchical representation of the image is computed once and, after each interaction, the hierarchy is reprocessed to obtain the segmentation.

In this work, we propose an interactive seeded WS cut algorithm within the framework of BPTs. First, we propose an algorithm to compute a seeded WS segmentation using the BPT associated to the input image. This procedure relies on two steps: 1) seeds propagation in the BPT and 2) connected components labeling. Second, we show how the seed propagation and the connected component labeling can be updated incrementally when the seeds are modified. Third, we propose a parallel algorithm to perform the connected component relabeling. The proposed methods are evaluated on several 2D and 3D images with real and simulated user interactions, showing how each contribution helps in reducing processing time and thus improving the user experience. Moreover, we perform run-time comparisons with state-of-the-art incremental and non-incremental watershed segmentation methods, showing a significant advantage for the proposed approach. A c++ implementation of the proposed methods is available at https://github.com/lebonq/incremental_watershed.

This work is an extension of a previous conference paper Lebon et al. (2023). Compared to this preliminary version, we provide: (i) a parallel algorithm for minimum spanning forest labeling, (ii) a more extensive experimental evaluation including 3D images, (iii) a comparison with state-of-the-art methods for images segmentation.

![Fig. 1. The three working areas: grayscale image, graph (in red), and BPT (in green). The dotted blue lines depict the bijection between non-leaf nodes and edges of the MST (in bold).](image)

2. Watershed cuts

This section reviews the notion of a WS cut, emphasizing its relation to minimum spanning trees and forests and the associated BPT datastructure that is used in the next section to propose a novel WS cut algorithm. WS cuts are deeply linked to MSTs Cousty et al. (2009): in the semi-supervised case, given an edge-weighted graph (e.g. representing the image) and a set of seed vertices (e.g., user-provided scribbles such that each scribble is superimposed to an object of interest to be segmented), a WS cut can be obtained as the cut induced by a minimum spanning forest such that each tree in the forest is rooted in one of the seeds Allène et al. (2010). The BPT is a tree-based hierarchical datastructure used to represent an MST. It allows one to efficiently browse MST’s edges. The BPT can be seen as a by-product of the efficient Kruskal’s MST algorithm. Figure 1 shows the BPT (in green) of an edge-weighted graph (in red) and its mapping (dotted blue) to the MST of this graph (bold edges). In particular, the leaves of the BPT are associated to the vertices of the MST and each non-leaf node is associated to an edge of the MST (this mapping is represented by the dashed segments in Figure 1). Intuitively, we can say that every non-leaf node of the BPT represents the addition of an edge to the MST during Kruskal’s algorithm where the added edge is used to merge the two connected regions that contain the edge extremities. An efficient WS cut algorithm based on BPT is presented in Cousty et al. (2013b); Najman et al. (2013) to handle the unsupervised case with no seed provided.

3. Semi-supervised watershed cut algorithm with interactions

This section introduces a new efficient algorithm to compute a WS cut from user-provided seeds, and to update its result based on user’s feedbacks given in the form of successive deletions and additions of seeds.

The overall workflow is presented in Figure 2. It comprises three main parts: 1) Computation of an MST and of an associated BPT; 2) Selection of the WS cut edges associated to the seeds provided by the user. These edges, which are found by browsing the BPT, correspond to the edges that must be deleted from the MST to obtain a minimum spanning forest rooted in the provided seeds. 3) Partitioning and labeling of the graph vertices according to the connected components (CC) of the forest obtained at step 2. This labeling is the resulting WS cut segmentation.

Step 1 is performed using the algorithm presented by Najman et al. (2013). An efficient algorithm for step 2 is given in Section 3.1. Section 3.2 discusses the labeling involved at step 3, and, finally, Section 3.3 shows that the proposed algorithm can be used to incrementally update the results of the workflow based on user’s feedback.

3.1. Tree-node marking

In this section, we present Algorithm 1 that returns the WS cut edges from an MST given in the form of a BPT datastructure and from a set of seed vertices. The algorithm works incrementally, seed by seed. For each seed, taken in an arbitrary order, one edge of the initial MST must be removed to cut the region of this seed from the regions of the current partition (obtained with the previously processed seeds). To find this edge, we search in the BPT for the lowest node that is both an ancestor of the new seed and an ancestor of an already processed seed. This node corresponds to the lowest edge of the MST that merges a region containing the new seed with a region containing an already processed seed. In order to “prevent” this merging and to cut the region of the new seed from the rest of
the partition, the MST edge associated to this node is tagged as
a "WS edge" and added to the set of edges to be deleted from
the MST.

Algorithm 1: ADD SEEDS

Data: $\mathcal{H}$: a BPT, seeds: a set of seeds and
visitCount;

Result: ws a set of edges to be removed and
visitCount updated.

1 $ws \leftarrow \emptyset$

2 foreach leaf $n$ of seeds do

3 \hspace{1em} while $n \neq \mathcal{H}.root$ and visitCount[$n$] \neq 2 do

4 \hspace{2em} $n := \mathcal{H}.parent[n]$ \hspace{1em}

5 \hspace{2em} visitCount[$n$] := visitCount[$n$] + 1 \hspace{1em}

6 \hspace{2em} if visitCount[$n$] = 2 then

7 \hspace{3em} \hspace{1em} $ws := ws \cup \{\mathcal{H}.mstEdge[p]\}$

To this end, we remove from the MST each WS edge returned
by Algorithm 1. Then, we perform a labeling of the CCs of the
forest with a Breadth First Search (BFS) algorithm.

3.3. Incremental workflow

The workflow presented in the previous sections is suited to
work in an incremental way, where one considers successive
addition or removal of seeds and the differential update of the
resulting WS cut and intermediary structures. Let us detail the
two cases corresponding to the addition and removal of a seed.

Seed addition: When new seeds are added, new WS edges
must be produced (one for each new seed). These new waters-
shed edges can be simply obtained, in a differential way, by a
call to Algorithm 1, provided that the array visitCount has
been memorized after the previous call and that the input data
only contains the new seeds. Then, we remove the returned new
WS cut edges from the current MSF and the connected compo-
nents split by the removal of these edges are relabeled using
breadth first search started at their extremity vertices. It can be
seen that the split of a component runs in time linear with
respect to the number of pixels in the split component.

Seed removal: When seeds are removed, WS edges must be
removed (one for each seed). We adapt Algorithm 1 to ob-
tain Algorithm 2 which accounts for seed removal in a dif-
ferential way. Such removal induces the merging of two re-
regions and the disappearance of a WS edge. The traversal
procedure is the same as for adding seeds except that for each par-
ent $p$, visitCount[$p$] is decremented and the parent browsing
stops if visitCount[$p$] = 1 after decrementation. Indeed, if
the value was decremented to 1, the current node is no longer
a WS node and the associated edge must be removed from the
set of WS cut edges. Regarding the associated minimum spanning forest and labeling, Algorithm 2 returns a set of WS cut edges that must be restored within the minimum spanning forest, inducing the merging of a CC for each of them. This is efficiently performed by constraining BFS to search and relabel only the smallest CC associated with one extremity of the edge. In this process the label of the larger CC is then spread on the smaller one. This can be done by keeping track of the size of each CC resulting in a linear time merging w.r.t. the number of vertices of the smallest CC.

As a result, this incremental workflow enables updating the segmentation in a time proportional to the number of pixels in the regions updated by the seed refinement.

Algorithm 2: REMOVE SEEDS

Data: $\mathcal{H}$: a BPT, seeds: a set of seeds and

Result: $\mathcal{S}$: a set of edges to be added and visitCount:

1. $\mathcal{S} \leftarrow \emptyset$
2. foreach leaf node $n$ of seeds do
3. if $n \notin \mathcal{H}.root$ and visitCount[$n$] $\neq 1$ then
4. visitCount[$n$] $\leftarrow$ visitCount[$n$] + 1
5. if visitCount[$n$] = 1 then
6. $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{H}.mstEdge[\text{parent}[n]]\}$

4. Parallelization

As shown in the experimental Section 5.3, most of the time spent to update a watershed cut segmentation in this differential framework is dedicated to CCs labeling (around 99.4% for 3D images). Therefore, to further improve the quality and speed of the interactions when dealing with large images, we investigate a parallel algorithm to reduce the time of this labeling by taking advantage of multicore CPUs.

The proposed method, described in Algorithms 3 and 4, is an improved version of the strategy proposed by Youkana et al. (2017) which is able to take into account that the considered CCs are trees which do not contain any cycle and that we can propagate the labels of the initial root vertices instead of the distances to the roots while performing the breadth-first search. The idea is to traverse iteratively the successive level sets found from the root vertices, each level set being traversed in parallel. Before this parallel traversal (Algorithm 3, Line 6), the current level set is partitioned (Line 5) in order to be distributed over the available processors in a balanced way. Since connected components are trees, the same vertex cannot be discovered from several distinct neighbors, thus from several distinct processors. Therefore, to check if a vertex is already labeled, we simply use the regular array $\text{label}$ without any synchronization.

Moreover, we also introduce two parameters in the method to control the degrees of parallelization and synchronization. The first one, denoted by MIN_BREADTH, allows the algorithm to dynamically switch from sequential to parallel search when the current level set (i.e. the propagation front) is large enough. The second parameter, denoted by PAR_DEPTH in Algorithm 3, allows us to control the number of parallel propagation steps performed between two synchronizations of the processors (union step performed at line 10), knowing that each synchronization between processors is time consuming, but that completely removing those synchronizations could lead to an unfair distribution of the data and an unbalanced workload between processors, resulting in a poor parallelization rate.

Algorithm 3: PARALLEL BREADTH-FIRST SEARCH

Data: $F$: a forest, roots: a set of root vertices to search from, each of which being in a distinct tree of $F$,
p: the number of processor;

Result: label: the updated segmentation.

1. $E \leftarrow \text{roots}$
2. $\text{label}[r]$: new unique label
3. while $E \neq \emptyset$ do
4. if $|E| > \text{MIN}_BREADTH$ then
5. $(E_1, \ldots, E_p) := \text{Partition}(E, p)$
6. foreach processor $i$ in $\{1, \ldots, p\}$ do in parallel
7. $S_i := \text{SuccessorLabeling}(F, E_i)$
8. foreach $v$ in $\{1, \ldots, \text{PAR}_\text{DEPTH}\}$ do
9. $S_i := \text{SuccessorLabeling}(F, S_i)$
10. $E \leftarrow \bigcup_{i=1}^{p} \{S_i\}$
11. else $E := \text{SuccessorLabeling}(F, E)$

Algorithm 4: SUCCESSORLABELING

Data: $F$: a forest, $E$: a set of vertices to explore;

Result: $S$: the set of explored vertices

1. $S \leftarrow \emptyset$
2. foreach vertex $v$ in $E$ do
3. foreach vertex $w$ adjacent to $v$ in $F$ do
4. if $\text{label}[w] \neq \text{label}[v]$ then
5. $S := S \cup w$
6. $\text{label}[w] := \text{label}[v]$

5. Experiments

We evaluate the method with three different experiments. The first one compares different methods with real user interactions. The second one relies on a publicly available dataset with a larger number of images and associated ground truth segmentation, for which the interactions are simulated by randomly selecting markers from the ground truth. The third is based on a 3D images dataset of CT scans. The parallel approach was only evaluated on 3D data. Other datasets were too small to demonstrate a speedup.

5.1. Experiment with user generated seeds

In this first experiment, we asked users to segment four images of size 2048 x 1536 pixels: three images come from the INRIA Holidays dataset Jegou et al. (2008) and one image is provided by ourselves. The users had to segment the images with a graphical interface to interactively edit seeds: they could draw green seeds for the object of interest and red seeds for
Table 1. Computation time (in milliseconds) of the different methods with user generated seeds. The second column indicates the initialization time, while the third and fourth columns show the average and maximum computation time for all user interactions. The last column represents the total computation time, which includes both the initialization and the sum of all user interactions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Init</th>
<th>Average</th>
<th>Max</th>
<th>Accumulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWS</td>
<td>210.0</td>
<td>9.2</td>
<td>89.3</td>
<td>514.1</td>
</tr>
<tr>
<td>NIWS</td>
<td>210.0</td>
<td>82.2</td>
<td>94.8</td>
<td>3164.4</td>
</tr>
<tr>
<td>DIFT</td>
<td>14.0</td>
<td>478.4</td>
<td>622.8</td>
<td>15886.3</td>
</tr>
<tr>
<td>OpenCV</td>
<td>∅</td>
<td>141.7</td>
<td>174.8</td>
<td>5089.4</td>
</tr>
<tr>
<td>Higra</td>
<td>∅</td>
<td>829.1</td>
<td>1239.4</td>
<td>29012.0</td>
</tr>
</tbody>
</table>

Fig. 4. Computation time of OpenCV (dotted orange), DIFT (plain purple), Higra (dashed red), NIWS (plain green) and IWS (dot-dash blue) during an interactive session (user generated seeds). The plain black curves show the number of pixels updated at each user interaction (right axis).

Fig. 5. Evolution of the segmentations thought user interactions. First, second and third show respectively the segmentation (yellow lines) after 2, half and all the user interactions. Red and green lines represent respectively the background and object seeds.

In this study, we consider two versions of the proposed method: (i) a non-incremental version (denoted NIWS) where we first compute the BPT and then, at each user interaction, we completely recompute the WS edges (using only Algorithm 1) and the induced labeling; and (ii) an incremental version (denoted IWS) where at each interaction we update visitCount (Algorithms 1 and 2) and the labeling by considering only the added/removed seeds. We also consider three state-of-the-art implementations of seeded WS, namely OpenCV Meyer (1992); Bradski (2000) (highly optimized library for image processing), Higra Perret et al. (2019) (generic library for hierarchical graph analysis) and another incremental method: the Differential Image Foresting Transform (DIFT) Falcao & Bergo (2004) (we used the authors’ implementation available at https://github.com/tvspina/ift-demo). The same sets of seeds were used for each tested method. The reported execution times are obtained using an Intel I7 13700H with 14 cores and 20 threads at 5.0GHz running GNU Linux.

The execution times are presented in Figure 4 and in Table 1. We see that the initialization cost (BPT creation) of both NIWS and IWS version of our method is quickly amortized during the segmentation process: IWS and NIWS have a much lower average execution time than other methods. In addition, we can see that the execution time of IWS is proportional to the number of pixels affected by seed updates as expected from the theoretical study. The upper bound is given by the first interaction, which labels all pixels (the first step is therefore equivalent to NIWS), and spikes in computation time occurs during mid- or end-interactions if the user updates seeds in a large CC, resulting in a significant number of pixel changes. Our results also indicate that there is no significant difference in computation time between adding or removing seeds.

5.2. Experiment with randomly generated seeds

To assess the methods on a standard dataset with more samples, we chose the BIG dataset Cheng et al. (2020) which is composed of large natural images (from 2048 by 1600 to 5000 by 3600 pixels). We use all 150 images and pair each image with a series of 70 seeds. Seeds are divided into two balanced...
Table 2. Computation times in nanoseconds per pixel on the BIG dataset with simulated interactions. The second column displays the initialization time for each method. The third and fourth columns show the average and maximum computation time for all interactions. The last column displays the total computation time, which includes the sum of the 70 interactions and the initialization time.

<table>
<thead>
<tr>
<th>Method</th>
<th>Init</th>
<th>Average</th>
<th>Max</th>
<th>Accumulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWS</td>
<td>88.1</td>
<td>6.8</td>
<td>27.6</td>
<td>563.6</td>
</tr>
<tr>
<td>NIWS</td>
<td>88.1</td>
<td>28.0</td>
<td>28.6</td>
<td>2051.3</td>
</tr>
<tr>
<td>DIFT</td>
<td>3.8</td>
<td>172.0</td>
<td>176.3</td>
<td>12044.9</td>
</tr>
<tr>
<td>OpenCV</td>
<td>0</td>
<td>60.9</td>
<td>61.2</td>
<td>4262.0</td>
</tr>
<tr>
<td>Higra</td>
<td>0</td>
<td>300.7</td>
<td>311.9</td>
<td>21045.3</td>
</tr>
</tbody>
</table>

Table 3. Computation times in nanoseconds per pixel on 3D volumes with simulated interactions. The second column displays the initialization time for each method. The third and fourth columns show the average and maximum computation time for all interactions. The last column displays the total computation time, which includes the sum of the 60 interactions and the initialization time.

<table>
<thead>
<tr>
<th>Method</th>
<th>Init</th>
<th>Average</th>
<th>Max</th>
<th>Accumulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWS</td>
<td>88.6</td>
<td>18.4</td>
<td>36.0</td>
<td>1189.6</td>
</tr>
<tr>
<td>IWS PAR</td>
<td>92.6</td>
<td>8.0</td>
<td>15.0</td>
<td>572.6</td>
</tr>
<tr>
<td>NIWS</td>
<td>85.6</td>
<td>37.7</td>
<td>38.5</td>
<td>2350.2</td>
</tr>
<tr>
<td>ITK</td>
<td>0</td>
<td>162.6</td>
<td>165.1</td>
<td>9754.6</td>
</tr>
</tbody>
</table>

In this experiment, we consider the same methodology as in Section 5.1. During the iterative process, seeds are alternatively picked within the object and within the background. In this experiment, only addition of seeds is considered, never removal.

The results are presented in Figure 6 and Table 2. For each interaction, the computation time is the average over all 150 images. With this experiment, the tendency observed in Section 5.1 is confirmed: the execution times of other methods are constant over the iterations whereas our incremental method shows decreasing execution times over the iterations, as the number of pixels affected by the interaction decreases. Furthermore, we observe that our proposed incremental methods significantly improves the response-time to user interactions compared to all other methods.

5.3. Experiment on 3D images

To assess the method on larger data, we considered the liver segmentation dataset Soler et al. (2010), which consists of 3D images acquired by CT scan. Out of them, we selected five 3D images of sizes from 512 × 512 × 124 to 512 × 512 × 260 voxels. We use the same methodology as in Section 5.2, using the liver masks as ground-truths. We generated 60 sets of seeds, each set containing 3D randomly positioned seeds of 100 pixels. Note that since our algorithm works on arbitrary graphs, no modifications are required to process the 3D images. For this experiment we tested the following four methods:

1. ITK: for each iteration we call the 3D watershed implementation available in Insight Toolkit (ITK) state-of-the-art library for medical imaging Beare & Lehmann (2006);
2. NIWS: the watershed-cut algorithm proposed in Section 3.1, thus without using the differential updates;
3. IWS: the watershed-cut algorithm proposed in Section 3.3, thus using differential updates but not using the parallel BFS;
4. IWS PAR: the watershed cut method proposed in Section 3.3 using the parallel BFS presented in Section 4. For this method, we experimentally chose to set the parameters MIN_BREADTH and PAR_DEPTH to 4000 and 3, respectively and 10 processors/threads were used.

The execution times of these methods are presented in Figure 7 and Table 3. The tendency observed in Section 5.2 and Section 5.1 is confirmed: the execution times for ITK and NIWS remain constant across iterations, whereas for the two differential methods, namely IWS and IWS PAR, the response time to user interactions is significantly reduced and furthermore decreases after few iterations, when the addition of new seeds starts to induce less changes in the segmentation results corresponding to small refinement on the results. On average, over all considered 3D images, ITK (resp. NIWS, IWS, and IWS PAR) requires 8 (resp. 1.8, 0.9, and 0.4) seconds per interaction, leading to a speedup of 20 when we compare the fastest IWS PAR method to the reference ITK. We also note that the use of parallelization in IWS PAR leads to a speedup of 2.3 over the IWS method.

In order to better analyze the proposed methods, we provide the time taken for the different steps of each method. Over all iterations, for NIWS, 95.7% (resp 0.66%, 3.64%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT), for IWS 92.3% (resp 0.5%, 7.2%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT), and for IWS PAR 82.8% (resp 1.03%, 16.17%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT). Moreover, if we consider only one interaction step, in average, for NIWS the time taken by the different steps remain the same whereas for IWS...
Fig. 7. Computation time in nano second per pixel on 3D images with simulated interactions. Our (IWS) is in blue, our paralleized version (IWS PAR) is in green, the non incremental version (NIWS) is in orange, ITK is in red. In order to avoid noisy visualization, the curve is smoothed by averaging values in a window of size 2.

6. Conclusion

We proposed an interactive seeded WS segmentation method that complies with an incremental process, exploiting causality within interactive sessions to achieve remarkable performance improvements and significantly enhance responsiveness. Additionally, we introduced a parallel method that leverages the properties of minimum spanning forest to further improve the responsiveness, especially for marker addition.

In future studies, we aim to investigate the optimal values for the parameters for parallel labeling to achieve a more significant speedup, particularly for a high number of processors/threads.

References


