# Algorithms and Implementation for Segmenting Tree Log Surface Defects

Van-Tho Nguyen[1], Bertrand Kerautret[2(✉)], Isabelle Debled-Rennesson[2], Francis Colin[1], Alexandre Piboule[3], and Thiéry Constant[1]

[1] LERFOB, AgroParisTech, INRA, 54000 Nancy, France
[2] LORIA, UMR CNRS 7503, Université de Lorraine, 54506 Vandœuvre-lés-Nancy, France
bertrand.kerautret@loria.fr
[3] ONF, RDI, 5 Rue Girardet, 54000 Nancy, France

**Abstract.** This paper focuses on the algorithms and implementation details of a published segmentation method defined to identify the defects of tree log surface. Such a method overcomes the difficulty of the high variability of the tree log surface and allows to segment the defects from the tree bark. All the algorithms used in this method are described in link to their source code which guarantees a full reproducible method associated to an online demonstration.

## 1 Overview of the Segmentation of Defects on Log Surface

In the computer imagery domain, the tubular objects are present in various applications of which segmentation and analysis are of most importance like in medicine (with the blood vessel segmentation [9,12]), biology (with the wood knot detections [11]) or industry [4,7]. In this work, we focus on the problem of wood quality estimation in relation to the presence of internal wood knots (Fig. 1). Since the latters have a structural link with defects on the surface of the trunk, the non destructive Terrestrial Laser Scanning (TLS) can be used to obtain an estimation of the wood quality. As described in previous work [14], different approaches were proposed following this strategy with for instance the fitting of primitive such as cylinder [10,17,18] or circle [19,20]. However, the existing approaches are not fully satisfactory in regards to the specificity of some species [20,21] or for the automation of the method [10,17,18]. Another limitation is the implementation details which are often missing and limit the reproduction of the methods.

A generic method to segment automatically defects on tree log surface that is robust to various tree species is still missing. In [14], we presented a novel approach to segment defects on tree logs from TLS data that is robust to different tested tree species in regard to bark roughness, longitudinal flexuosity or cross-sectional ovality. However, being limited on the number of pages, we were not able to provide enough details about algorithms, implementations and also how


B. Kerautret et al. (Eds.): RRPR 2016, LNCS 10214, pp. 150–166, 2017.
DOI: 10.1007/978-3-319-56414-2_11

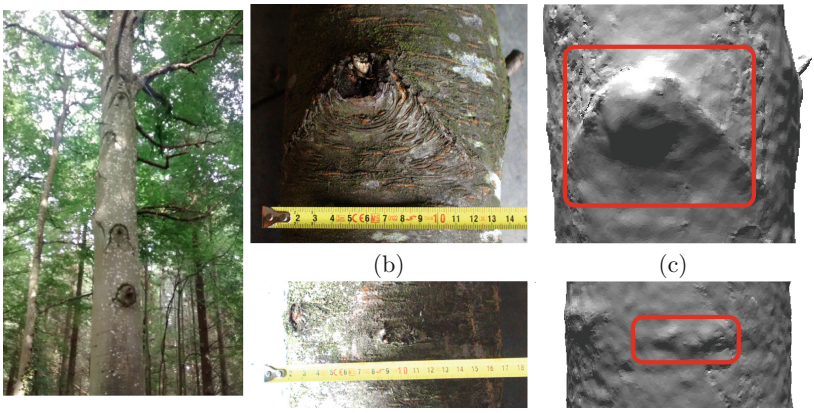


**Fig. 1.** (a) Defects on a standing tree in forest. (b,c) A Chinese mustache and its corresponding mesh. (d,e) A small defect in ellipse shape and its corresponding mesh.

to reproduce the best results. As a complement of [14], this paper aims to provide the implementation details of the main algorithms and to discuss the choice of parameter values and the reader can also access to the source code and other resources to reproduce the published results.

Our approach to segment tree log surface defects consists of four main steps (see Fig. 2(a)). In the first step, we compute the centerline of the tree log by the accumulation of surface normal vectors. In the second step, based on this centerline, we can compute the distance to the centerline and then convert the point cloud from the cartesian coordinate system to cylindrical coordinate system. The third step concerns the computation of the reference distance to the centerline using a patch of neighbouring points. Finally, in the fourth step we compute the difference between the reference and the real distances to the centerline before applying an automatic threshold to binarize the point cloud according to the statistical distribution of the values.

The following section describes the details and implementations of our algorithms. The Sect. 3 describes the steps to reproduce the results. We discuss how to choose the best values of the most important parameters and show some limitations of our method in the Sect. 4.

## 2 Algorithms and Implementations

In this section, we describe in detail our algorithms and its implementations. The program is written in *C++* using libraries DGtal [1] for the normal accumulation and the visualization, PCL [16] for the kd-tree, GNU Scientific Library (GSL) [6] for the smoothing and interpolation of the centerline by cubic Spline. Figure 3 shows main classes and methods that will be described in detail in this section.

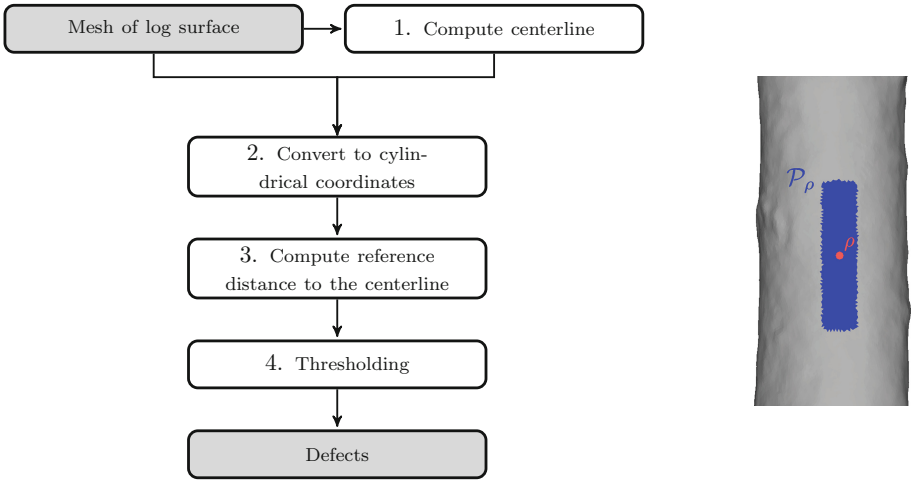



**Fig. 2.** (a) Overview of the algorithm. (b) Illustration of a patch $\mathcal{P}_\rho$ in blue which is used to estimate the reference distance to the centerline for the orange point. (Color figure online)

By convenience, our source files have the same name as class names. For example the class `Centerline` is implemented in the file *Centerline.h* and *Centerline.cpp*.

### 2.1 Compute Centerline

In the proposed method, the centerline plays an important role because the key variable is its distance to the input points. Thus, the centerline must be precisely assessed. Our method to compute the centerline is based on the surface normal accumulation proposed in [7] with modifications optimized to tree log data. The implementation of this method is simple (Fig. 4(a)) and consists of four steps: (i) accumulation of surface normal vector, (ii) tracking of the centerline, (iii) optimization by elastic forces (see Algorithm 1) and (iv) BSpline interpolation.

**(i) Accumulation of surface normal vectors.** This step consists in computation of a volumetric image of which voxel value stores score of the accumulation (Fig. 4(b)) which is computed as follows. Starting from a mesh face $f_i$ with its normal vector $\overrightarrow{n_i}$ and applying a normal oriented directional scan along a distance $d_{acc}$ (oriented toward the object interior). During this scan, we increase by one all voxels of the digitized space which are crossed by the 3D line defined from the face center and from the direction $\overrightarrow{n_i}$ (see Fig. 4(a,b)) and Algorithm 1 of [7] for more details).

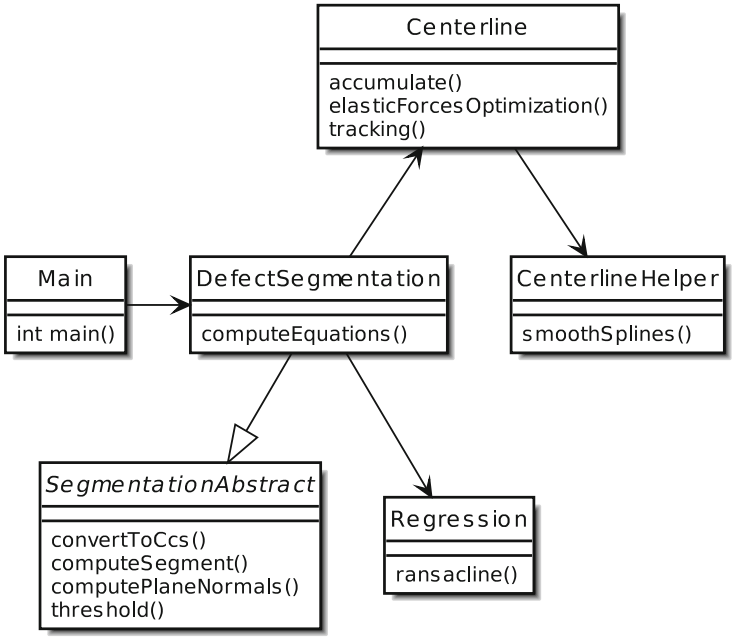



**Fig. 3.** Diagram class of our implementation.

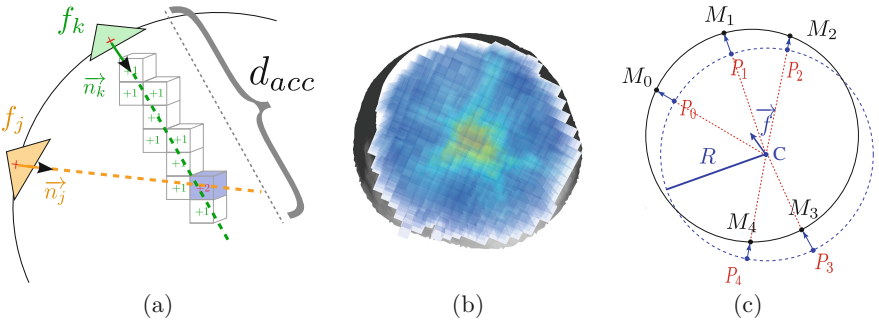
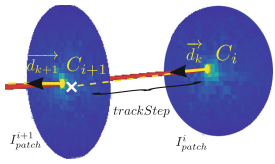



**Fig. 4.** Illustrations of the main ideas of the centerline method based on surface normal accumulation: (a) scan from mesh faces along a distance $d_{acc}$, (b) resulting accumulation score. (c) illustrates the step of elastic force optimization.

**(ii) Tracking the centerline.** Based on the result of previous algorithm, the centerline curve is obtained by a simple tracking process by looking at the local maxima accumulation points $C_i$ of 2D patch image resulting from the projection in the $\overrightarrow{d_k}$ direction (see following figure). This direction $\overrightarrow{d_k}$ is computed from



all the points which are at the origin of the accumulation value (see Algorithm 2 of [7] for more details).

**(iii) Elastic force optimization.** Due to the digital space and the mesh quality, the normal vectors may not be perfectly convergent and some irregularities on the resulting centerline can appear. As suggested in [7], we improve this step by using an optimization Algorithm 1. The main idea is to minimize the error $E_s(C)$ defined as the sum of the squared difference between the mean radius $\bar{R}$ of the log and the norm of the projection vector formed by log center $C_i$ and its associated input mesh points $M_i$ onto the plane perpendicular to the direction vector $\overrightarrow{d_i}$ (see Fig. 4(c)).

**(iv) Spline Interpolation.** The purpose of the interpolation is to obtain a smoother centerline. We firstly choose eight points on the centerline based on the curvature. The points with a too strong curvature change are eliminated by testing the angle $\phi$ defined by the considered point, and its two immediate neighbors on both sides along the centerline. If $\phi$ is smaller than $3\pi/4$ then the point is eliminated. Finally, a cubic spline interpolation is applied to these chosen points. Our implementation of this step is based on cubic spline which is implemented in GNU scientific library [6]. Figure 5(a) shows the final result of a computed centerline.

### 2.2 Convert to Cylindrical Coordinates

The purpose of this conversion is to simplify the search of neighbors. Firstly, we divide the point cloud into slices by the centerline segment computed in the previous section. Each point in the point cloud belongs to only one slice (Algorithm 2). As shown in [14], we define a local coordinate system $O_i x_i y_i z_i$ for each slice (see Algorithm 4) where $\{\overrightarrow{u_i}, \overrightarrow{v_i}, \overrightarrow{w_i}\}$ are basis vectors with the origin $O_i$ (i.e. the first point on the segment of the centerline corresponding to the slice). $\overrightarrow{w_i}$ is defined as the segment $C_i C_{i+1}$. The axis $O_0 y_0$ can be arbitrary and we choose $O_0 y_0 = Oy$. With $i > 0$, $\overrightarrow{v_i}$ is computed as follows. Let $\overrightarrow{n_i}$ be the normal vector of the plane formed by $\overrightarrow{w_i}$ and $\overrightarrow{v_{i\text{-}1}}$: $\overrightarrow{n_i} = \overrightarrow{w_i} \times \overrightarrow{v_{i\text{-}1}}; i \in [1, m-1]$. $\overrightarrow{v_i}$ is the cross product of $\overrightarrow{w_i}$ and $\overrightarrow{n_i}$. And finally, $\overrightarrow{u_i}$ is computed by the cross product of $\overrightarrow{v_i}$ and $\overrightarrow{w_i}$. Figure 5(b) shows the computation of coordinate system for a point from its Cartesian coordinates.

### 2.3 Compute Reference Distance to the Centerline

To compute the reference distance to the centerline of the point $\rho$, we need to query a narrow patch $\mathcal{P}_\rho$ containing the neighbors of $\rho$ (see Fig. 2(b)). The patch size is defined by two parameters $\varphi$ and $\tau$ with $\varphi = l/\bar{r}$ where $\tau$ (resp. $l$) is the length (resp. arc length) of the patch and $\bar{r}$ is the mean of the distance to the centerline of all points. More formally the patch $\mathcal{P}_\rho$ can be defined as:

$$\mathcal{P}_\rho = \{\rho_j \mid |\theta_{\rho_j} - \theta_\rho| \leq \frac{\varphi}{2}, |z_{\rho_j} - z_\rho| \leq \frac{\tau}{2}\} \tag{1}$$

**Algorithm 1. elasticForcesOptimisation**: Optimize the centerline by elastic forces

```
   Data: rawCenterline // Raw centerline, output of Algorithm 2 of [7]
   Data: mesh // Input mesh to recover faces associated to centerline points
   Output : optimizedCenterline
   Variable: epsilon //Error
 1 //store associated faces in the 2D patches
 2 mapFaces = emptyMap()
 3 sumradii=0
 4 foreach point in rawCenterline do
 5     faces = get2DpatchImage(point, mesh)
 6     mapFaces[point] = faces
 7     foreach face in faces do
 8         centerPoint = face.center()
 9         vectorNormal = face.getVectorNormal()
10         vectorRadial = centerPoint - point
11         sumradii = vectorRadial.norm()
12 meanRadii = sumradii / mesh.nbFaces()
13 DeltaError = infinity
14 previousTotalError
15 first=true
16 while DeltaError > epsilon do
17     totalError = 0.0
18     for i = 0 to rawCenterline.size() - 1 do
19         point = optimizedCenterline[i]
20         faces = mapFaces[point]
21         sumForce = {0,0,0}
22         count=0
23         foreach face in faces do
24             centerPoint = face.center()
25             vectorNormal = face.getVectorNormal()
26             radialVector = centerPoint - point
27             alpha = anglebetween(vectorNormal, radialVector)
28             if alpha ¿ π/6 then
29                 continue
30             forceMagnitude = radialVector.norm() - meanRadii
31             force = radialVector.normalized()*forceMagnitude
32             totalError += forceMagnitude*forceMagnitude
33             sumForce += force
34             count++
35         //project of sumForces to normal vector
36         direction = dirImage[rawCenterline[i]].normalized()
37         sumForcesDir = direction.dot(sumForces)/vectDir.norm()/vectDir.norm()*vectDir;
38         radialForces = sumForces - sumForcesDir;
39         if count > 0 then
40             optimizedCenterline[i] += radialForces/count
41     if first then
42         DeltaError = totalError first = false
43     else
44         DeltaError = abs(totalError - previousTotalError)
45     previousTotalError = totalError - previousTotalError
46 return optimizedCenterline
47 .
```

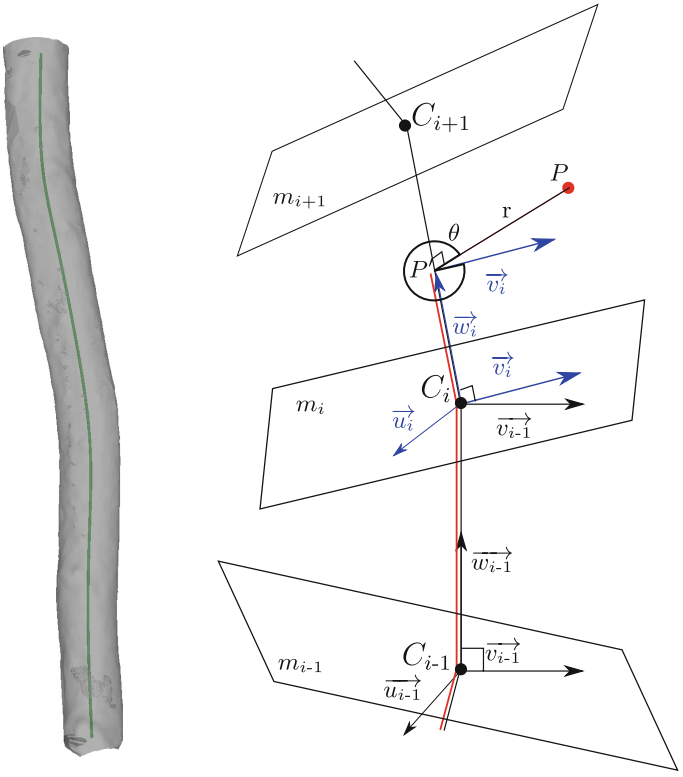



**Fig. 5.** (a) Centerline of a Beech log. (b) Computation of the coordinate system $\rho(r, \theta, z)$ from the point $P$ with $z$ is the red line. (Color figure online)

**Algorithm 2. computePlanes**: Compute normal vector of planes that divide the log into slices

**Data:** Centerline $\mathcal{C} = \{C_0, C_1, \ldots, C_m\}$
**Result:** List of normal vector m

```
m = emptySet()
```

$$m[0] = \frac{\overrightarrow{C_0C_1}}{\left\|\overrightarrow{C_0C_1}\right\|}$$

**for** $i = 1$ *to* $m - 1$ **do**

$$m[i] = \frac{\overrightarrow{C_{i-1}C_i} + \overrightarrow{C_iC_{i+1}}}{\left\|\overrightarrow{C_{i-1}C_i} + \overrightarrow{C_iC_{i+1}}\right\|}$$

**end**

**return** m

Because the number of points may be large, we use a kd-tree to speedup the query. We use the kd-tree implemented in the *Flann* library [3,13] which is included in the *PCL* library [16]. This library provides a range base query, so

**Algorithm 3. getSegmentId**: Compute the corresponding segment of a point

**Data:** Point $P$
**Data:** Centerline $\mathcal{C} = \{C_0, C_1, \ldots, C_m\}$
**Result:** segmentId
$lastSign \leftarrow 1$
**for** $i = 1$ *to* $m$ **do**
  $sign = \overrightarrow{C_iP}.\overrightarrow{ns[i]}$;
  **if** *sign * lastSign ¡= 0* **then**
    **return** i - 1;
  **end**
  lastSign = sign;
**end**
**return** m -1

**Algorithm 4. convertToCcs**: Convert the point cloud to cylindrical coordinate system

**Data:** Point cloud $\mathcal{P}_d = \{P_0, P_1, \ldots, P_n\}$
**Data:** Centerline $\mathcal{C} = \{C_0, C_1, \ldots, C_m\}$
**Data:** Local coordinate systems $\mathcal{L} = \{O_0x_0y_0z_0, O_1x_1y_1z_1, \ldots, O_{m-1}x_{m-1}y_{m-1}z_{m-1}\}$
**Result:** Point cloud in cylindrical coordinate system
**for** $i = 1$ *to* $n$ **do**
  $s = getSegmentId(P_i)$ //Algorithm 2
  $\overrightarrow{d} = \frac{\overrightarrow{C_sC_{s+1}}}{\left\|\overrightarrow{C_sC_{s+1}}\right\|}$ //projection of $P_i$ onto segment s of the centerline
  $P' = \overrightarrow{d}.\overrightarrow{C_sP_i}$;
  $r_i = \left\|\overrightarrow{P'P_i}\right\|$
  **if** *s ¿ 0* **then**
    $z_i = \sum_{j=1}^{s} \left\|\overrightarrow{C_{j-1}C_j}\right\| + \overrightarrow{C_jP_i}.\overrightarrow{d}$
  **else**
    $z_i = \overrightarrow{C_jP_i}.\overrightarrow{d}$
  **end**
  $\theta = \frac{\arccos \overrightarrow{P'P}.\overrightarrow{v_s}}{\left\|\overrightarrow{P'P}\right\|}$
  //correction of angle
  $\overrightarrow{t} = \overrightarrow{v_s} \times \overrightarrow{d}$
  **if** $\overrightarrow{t}.\overrightarrow{P'P} < 0$ **then**
    $\theta = 2\pi - \theta$;
  **end**
**end**

that at the point $\rho$, we query all the neighbors located inside the sphere of center $\rho$ and $radius = \frac{\tau}{2}$:

$$\mathcal{Q}_\rho = \{\rho_j \mid \left|z_{\rho_j} - z_\rho\right| \leq \frac{\tau}{2}\} \tag{2}$$

The return must be refined to get the patch:

$$\mathcal{P}_\rho = \{\rho_j \in \mathcal{Q} \mid \left|\theta_{\rho_j} - \theta_\rho\right| \leq \frac{\varphi}{2}\} \tag{3}$$

Note that the query of patch can be improved by using a kd-tree for the $\theta$ and $z$ coordinates in the cylindrical coordinate system with a regard of the circular problem of $\theta$.

To compute the reference distance to the centerline (denoted $\hat{r}$) of the point $i$, we consider the profile of $r$ by $z$ and then compute a RANSAC [5] based linear regression (i.e. $\hat{r} = az + b$, see Fig. 6).

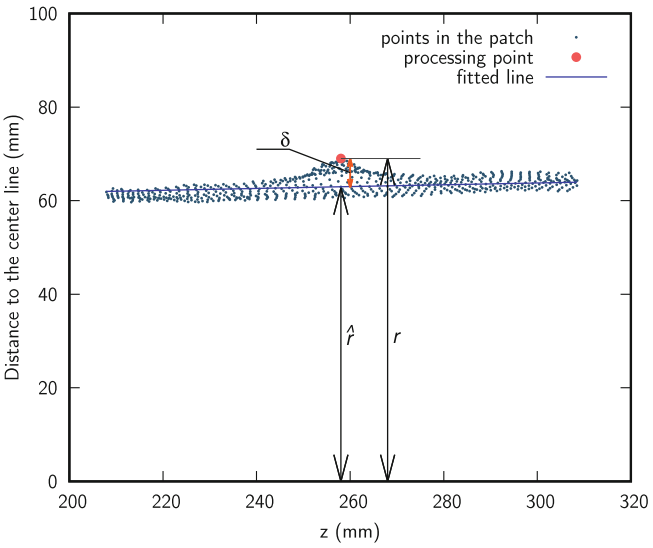



**Fig. 6.** Computation of the reference distance to the centerline for a given point.

### 2.4 Thresholding

After computing the reference distance to the centerline of points in the point cloud, we use the unimodal thresholding method proposed by Rosin in [15] to binarize the point cloud. Firstly, at each point $\rho$, we compute the difference between the distance to the centerline $r$ and the reference distance to the centerline $\hat{r}$ for all points ($\delta_\rho = r_\rho - \hat{r_\rho}$).

Then, we compute the histogram of $\delta_\rho$ of all points in the cloud. The algorithm firstly finds the bin with maxima frequence ($B_1$) and then finds the first null bin ($B_2$) at the right size of the histogram. Finally, the algorithm loops over the bins situated between $B_1$ and $B_2$ and chooses the bin that maximizes the distance to the line $B_1B_2$ (Fig. 7). The Algorithm 5 shows the details of the implementation.

**Algorithm 5. threshold**: Compute threshold by the Rosin method

**Data:** $\mathcal{D} = \{\delta_0, \delta_1, \ldots, \delta_n\}$
**Data:** $binSize$
**Result:** threshold T

```
nbBins = (max(D) − min(D)) / binSize
histogram = array[nbBins]
for i = 1 to n do
    binId = (ρ_i − min(D)) / binSize
    histogram[binId]++
end
maxFrequence = max(histogram)
maxFrequenceIndex = indexOf(maxFrequence)
nullFrequenceIndex = nbBins -1
for i = maxFrequenceIndex to nbBins do
    if histogram[i] == 0 then
        nullFrequenceIndex = i
        break
    end
end
bestDist = 0
bestDistIndex = maxFrequenceIndex
for i = maxFrequenceIndex to nullFrequenceIndex do
    AB = line({maxFrequenceIndex, maxFrequence}, {nullFrequenceIndex, 0})
    if distance(i, histogram[i], AB) ¿ bestDist then
        bestDist= distance(i, histogram[i], AB)
        bestDistIndex = i
    end
end
return T = bestDistIndex * binSize + min(D)
```

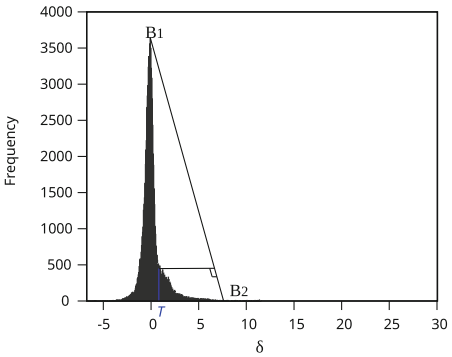



**Fig. 7.** Computation of the threshold $T$. Due to the different scales on the two axis, the two perpendicular lines are not shown correctly.

## 3 Reproducing the Results

The source code used to generate the present results is given at the following *GitHub* repository:

https://github.com/vanthonguyen/treelogdefectsegmentation

and an online demonstration is available at the following url:

http://ipol-geometry.loria.fr/~kerautre/ipol_demo/TDD_IPOLDemo/

The command to reproduce the results is:

```
segmentation -i mesh.off --voxelSize 5 --accRadius 100 --trackStep 20 \
--patchWidth 25 --patchHeight 100 --binWidth 5 --invertNormal true -o prefix
```

The program accepts the following parameters:

- `voxelSize`: the voxel size which is related to the resolution of the point cloud. In our experiments, the density of the point cloud is about 25 points per $cm^2$, we have chosen the voxel size as the squared root of the density which is 5.
- `accRadius`: the radius of the normal accumulation which should be greater than the real radius of the log.
- `trackStep`: the distance between two points in the tracking of the centerline.
- `patchWidth`, `patchHeight`: the width and height of the patch in the search of neighbors. The choice of the patch size must guarantee that the height is several time greater than the width and that the width is enough large to avoid an empty patch. In our experiments, we fixed the width equals to 25 mm and the height equals to 100 mm.
- `binWidth`: the width of histogram bins used to compute the threshold by the Rosin method.
- `invertNormal`: used when the direction of normal vectors is outside of the object.
- `output`: the prefix of output files. The program writes the output on some files: (1) output mesh with highlighted defects in green (*prefix-defect.off*), (2) the distance map of $\delta$ (*prefix-error.off*), (3) the face ids of defects (*prefix-def-faces.id*) which can be used with the tool *colorizeMesh* (included in source code), (4) the point ids of defects (*prefix-defect.id*) which can be used to compare with the ground truth. The format of these two last files are simply a list of integers separated by newlines.

The results obtained on different tree species including ground truth error measures were already presented in previous work [14]. We focus in this part on the reproduction of the previous results and in the experimental stability measure of the different parameters.

**Reproducing results on various species.** From the previous command line, the method can be applied directly to different species without the need to change the default parameters. The Fig. 8 presents some results generated from the example directory of the *GitHub* repository. All the results were generated with the same default parameters: `voxelSize` = 5 mm, `accRadius` is chosen by 1.5 time the maximum radius of the log (by default set to 200), the `patchWidth` = 25 mm, `patchHeight` = 100 mm. Note that the similar results are available on the demo site [2] with other examples.

**Reproducing ground-truth comparisons.** The Fig. 9 illustrates the comparisons of the previous results with the ground truth constructed by INRA experts

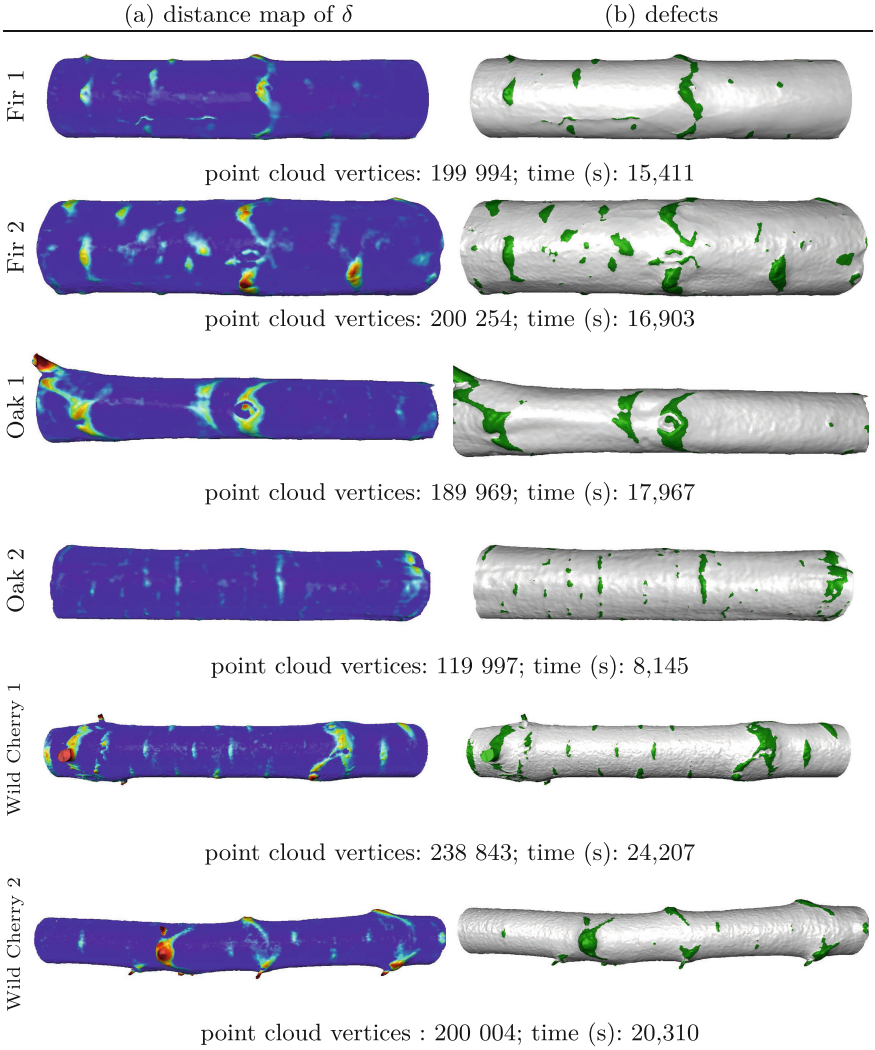
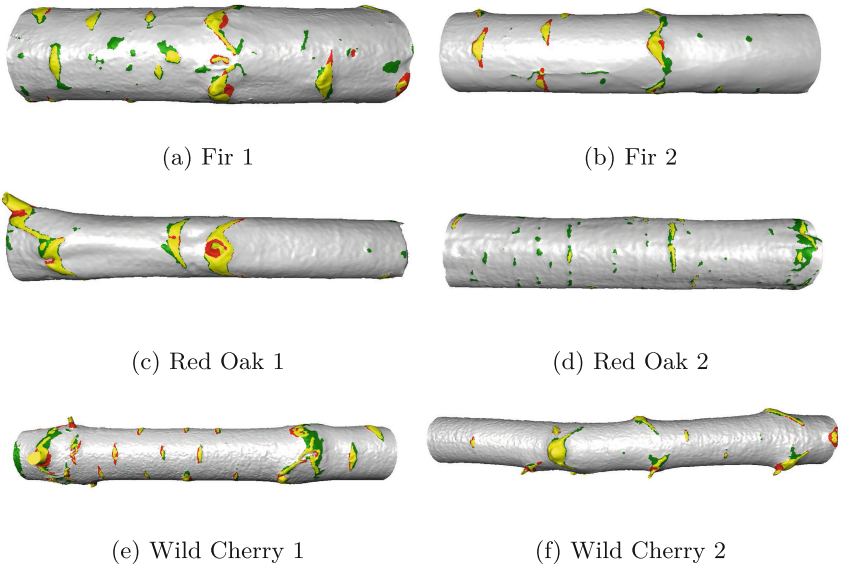



**Fig. 8.** Visualization of the distance map of $\delta$ (column (a)) and defects (column (b)) obtained on various species with the same default parameters (`voxelSize` = 5, `accRadius` = 200, `trackStep` = 20, `patchWidth` = 25, `patchHeight` = 100, `binWidth` = 0.01). The execution time were obtained on a *MacOS* 2,5 GHz *Intel Core i7*.

built from logs of different tree species and diameters, with a one meter length. Such comparisons can also be reproduced from the following command line:

```
colorizeMesh -i examples/WildCherry1.off -r examples/WildCherry1-grountruth.id
-t res_WildCherry-def-faces.id  -o compareWildCherry.off
```

To ensure reproducibility the ground-truth files are also given in the *GitHub* repository.



(a) Fir 1 (b) Fir 2

(c) Red Oak 1 (d) Red Oak 2

(e) Wild Cherry 1 (f) Wild Cherry 2

**Fig. 9.** Segmentation results of the proposed method for some logs: the detected zones, overlaying the ground truth are in yellow, non overlaying zones are in green, and the non-detected zones are in red. (Color figure online)

**Influence of the parameters.** We experimented the influence of some parameters in order to measure their impact towards the result quality. First, we measure the influence of the patch size when the recommended criteria are not observed. In the experiments of Fig. 10, the patch size has been changed with various values and the results appear visually robust. The most significant quality variations are visible only when using not recommended values (cases of too small (resp. big) size (images (f) resp. (d) of Fig. 10) or in case of bad orientations of the patch (images (e) of Fig. 10)). In a second time, the robustness of the Rosin automatic thresholding method [15] was experimented by changing its inside parameters (`binWidth`). As shown on the experiments of the Fig. 11, the change of this parameter is not very sensitive (see images (a–d)). Finally, the change of multiple default parameters also show small quality variations (see Fig. 11(e,f)).

## 4 Discussions

The previous results show that our method can precisely segment the defects and seems to be robust to different tree species or to geometrical variations. It must be preferred to the cylindrical-based method. The actual limitations appear when protruding branches are present on the log surface. To overcome this configuration, we envisage to use an additional method to segment branches before applying the proposed algorithms.

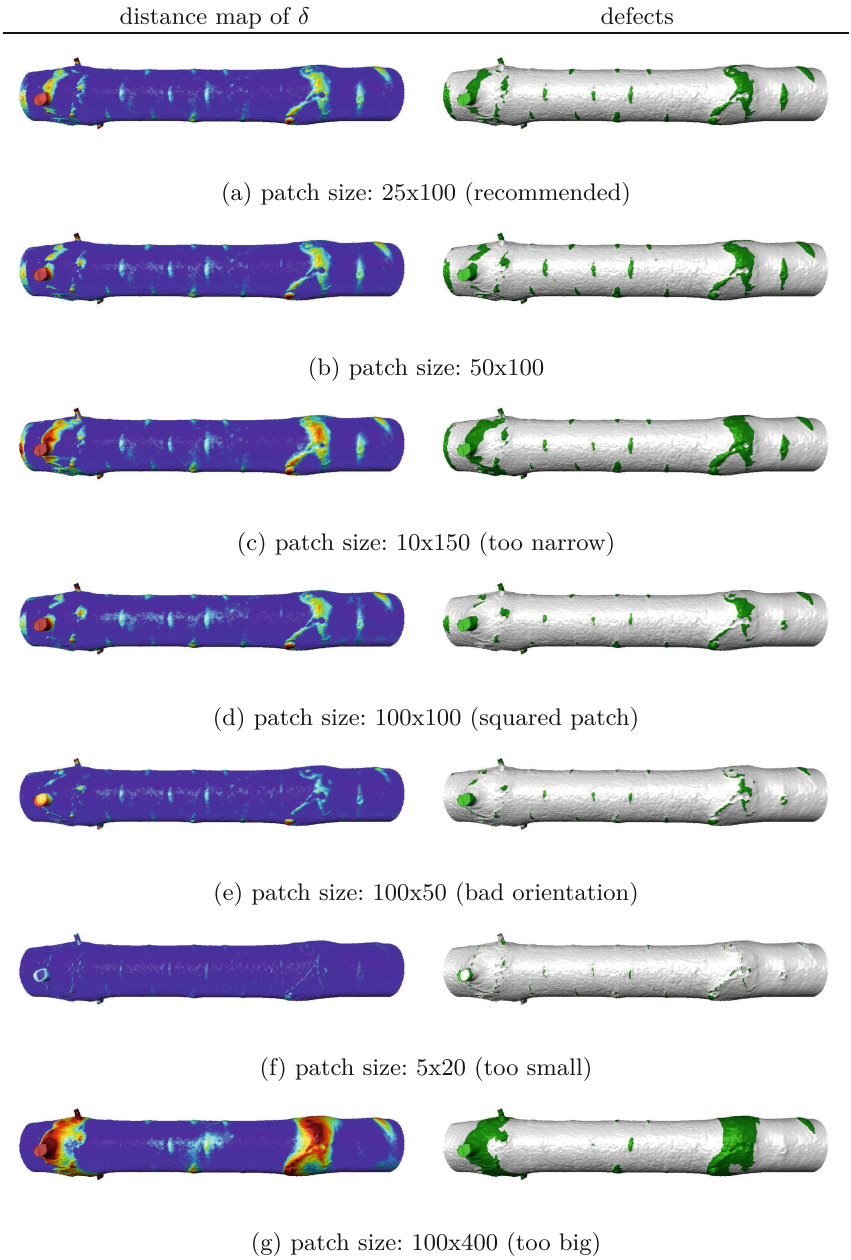
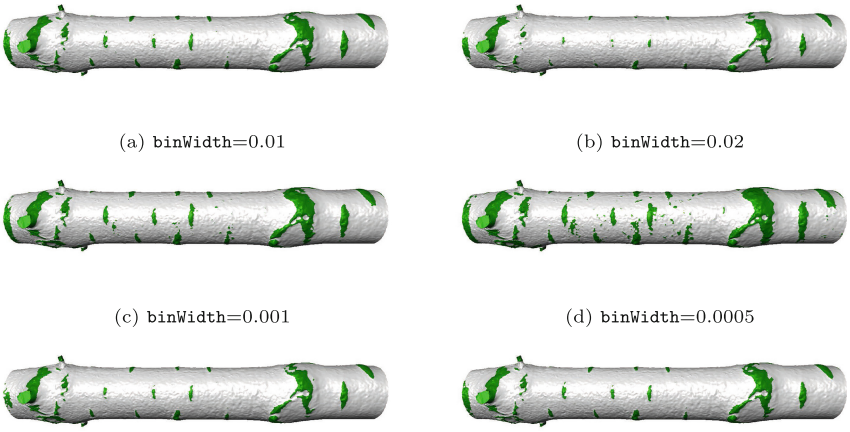



**Fig. 10.** Experimentation of the method stability towards the patch size parameters (`patchWidth`, `patchHeight`).



(a) binWidth=0.01 (b) binWidth=0.02

(c) binWidth=0.001 (d) binWidth=0.0005

(e) accRadius=300, voxelSize=3, trackStep=10 (f) accRadius=100, voxelSize=10, trackStep=25

**Fig. 11.** Illustration of the influence of the `binWidth` parameter (a–d). Images (e,f) illustrate the stability by changing several parameter values.

We have also observed some limit cases for the tracking algorithm used to compute the centerline which could be sensible to the voxel size and to the noisiness of the input mesh. In particular, if the voxel size is too large or if the input mesh is too noisy, during the tracking process, a point of the centerline might be located in an empty voxel (voxel with null accumulation) and the tracking process may be terminated prematurely. Even if this case can be resolved by a local search of non null accumulation, in future work we plane to use the tracking method based on the confidence of accumulation [8]. As presented in the previous section, the choice of voxelSize is crucial. If the voxel size is smaller than the resolution of the data, the program may require more memory and time with no better result.

The width of bin used in the thresholding method (`binWidth`) should be small for a more precise threshold especially when one want to detect the small defects. For this reason, in our experimentations, the `binWidth` was fixed to 0.01 mm.

The choice of the patch size must guarantee that the height is several times greater than the width and enough large to avoid empty patch. Moreover the patch height should be greater than the largest defect height. In our experimentations, we fixed the width equals to 25 mm and the height equals to 100 mm.

## 5 Conclusion

This paper has presented the implementation details of our novel algorithms to segment the defects on the surface of tree logs from Terrestrial Laser Scanning data. The proposed method consists in algorithms to precisely compute

the centerline of tree logs and algorithms to compute the reference distance to the centerline and to threshold the point cloud. The experiments showed that the method could precisely segment defects with complex shape like the Chinese mustaches and small defects. The actual limit case are the logs with very prominent defects like a living branch for which the proposed method did not have a good performance. All the results presented in this paper are reproducible both from the source code or from the online demonstration.

**Acknowledgment.** This work was supported by the French National Research Agency through the Laboratory of Excellence ARBRE (ANR-12- LABXARBRE-01) and by the Lorraine French Region.

## References

1. DGtal: Digital Geometry tools and algorithms library. http://dgtal.org
2. Online demonstration. http://ipol-geometry.loria.fr/~kerautre/ipol_demo/TrunkDefaultMeasure
3. Flann library, October 2016
4. Bauer, U., Polthier, K.: Generating parametric models of tubes from laser scans. Comput. Aided Des. **41**(10), 719–729 (2009)
5. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
6. Gough, B.: GNU Scientific Library Reference Manual. Network Theory Ltd., London (2009)
7. Kerautret, B., Krähenbühl, A., Debled-Rennesson, I., Lachaud, J.-O.: 3D geometric analysis of tubular objects based on surface normal accumulation. In: Murino, V., Puppo, E. (eds.) ICIAP 2015. LNCS, vol. 9279, pp. 319–331. Springer, Cham (2015). doi:10.1007/978-3-319-23231-7_29
8. Kerautret, B., Krahenbl, A., Debled Rennesson, I., Lachaud, J.O.: Centerline detection on partial mesh scans by confidence vote in accumulation map. In: Proceedings of ICPR 2016 (2016). To appear
9. Kirbas, C., Quek, F.: A review of vessel extraction techniques and algorithms. CSUR **36**(2), 81–121 (2004)
10. Kretschmer, U., Kirchner, N., Morhart, C., Spiecker, H.: A new approach to assessing tree stem quality characteristics using terrestrial laser scans. Silva Fenn **47**, 14 (2013)
11. Krhenbhl, A., Kerautret, B., Debled-Rennesson, I., Mothe, F., Longuetaud, F.: Knot segmentation in 3D CT images of wet wood. Pattern Recogn. **47**(12), 3852–3869 (2014)
12. Lesage, D., Angelini, E.D., Bloch, I., Funka-Lea, G.: A review of 3D vessel lumen segmentation techniques: models, features and extraction schemes. Med. Image Anal. **13**(6), 819–845 (2009)
13. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2227–2240 (2014)
14. Nguyen, V.T., Kerautret, B., Debled-Rennesson, I., Colin, F., Pipoule, A., Constant, T.: Segmentation of defects on log surface from terrestrial lidar data. In: Proceedings of ICPR 2016 (2016). To appear

15. Rosin, P.L.: Unimodal thresholding. Pattern Recogn. **34**(11), 2083–2096 (2001)
16. Rusu, R.B., Cousins, S.: 3D is here: point cloud library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011
17. Schütt, C., Aschoff, T., Winterhalder, D., Thies, M., Kretschmer, U., Spiecker, H.: Approaches for recognition of wood quality of standing trees based on terrestrial laserscanner data. ISPRS **36**, 179–182 (2004)
18. Stängle, S.M., Brüchert, F., Kretschmer, U., Spiecker, H., Sauter, U.H.: Clear wood content in standing trees predicted from branch scar measurements with terrestrial lidar and verified with X-ray computed tomography 1. Can. J. For. Res. **44**(2), 145–153 (2013)
19. Thomas, L., Mili, L.: A robust GM-estimator for the automated detection of external defects on barked hardwood logs and stems. IEEE Trans. Signal Process. **55**(7), 3568–3576 (2007)
20. Thomas, L., Shaffer, C.A., Mili, L., Thomas, E.: Automated detection of severe surface defects on barked hardwood logs. For. Prod. J. **57**(4), 50 (2007)
21. Thomas, L., Thomas, R.E.: A graphical automated detection system to locate hardwood log surface defects using high-resolution three-dimensional laser scan data. In: 17th Central Hardwood Forest Conference. vol. 78, p. 92 (2010)