Fast Local Thickness

Vedrana Andersen Dahl and Anders Bjorholm Dahl
Technical University of Denmark
{vand, abda}@dtu.dk

Abstract

We propose a fast algorithm for the computation of local thickness in 2D and 3D. Compared to the conventional algorithm, our fast algorithm yields local thickness in just a fraction of the time. In our algorithm, we first compute the distance field of the object and then iteratively dilate the selected parts of the distance field. In every iteration, we employ small structuring elements, which makes our approach fast. Our algorithm is implemented in Python and is freely available as a pip-installable module. Besides giving a detailed description of our method, we test our implementation on 2D images and 3D volumes. In 2D, we compute the ground truth using the conventional local thickness methods, where the distance field is dilated with increasingly larger circular structuring elements. We use this as a reference to evaluate the quality of our results. In 3D, we have no ground truth since it would be too time-consuming to compute. Instead, we compare our results with the golden standard method provided by BoneJ. In both 2D and 3D, we compare with another Python-based approach from PoreSpy. Our algorithm performs equally well or better than other approaches, but significantly faster.

1. Introduction

Contemporary high-resolution 3D microscopy scanners typically produce volumes containing 2048^3 voxels, or even 4096^3 voxels. With volumes containing billions of voxels, even the simplest processing methods may be computationally demanding. This is especially pronounced when processing involves using a kernel that also grows cubically in size. And processing large images often requires the use of large kernels. For this reason, efficient 3D analysis requires re-visiting processing algorithms. In this paper, we consider the computation of the local thickness.

For an object in 3D, the local thickness at any point of the object is defined as the radius of the largest sphere which fits inside the object and contains the point, see Fig. 1. Despite this simple definition, computing local thickness using a direct implementation of the definition is computationally demanding, and for large volumetric data computationally infeasible. The time complexity of the conventional local thickness algorithm is such that if it takes 1 second to process a volume of size 512^3, processing a 2048^3 volume takes 4.5 hours and a 4096^3 volume takes 24 days to process.

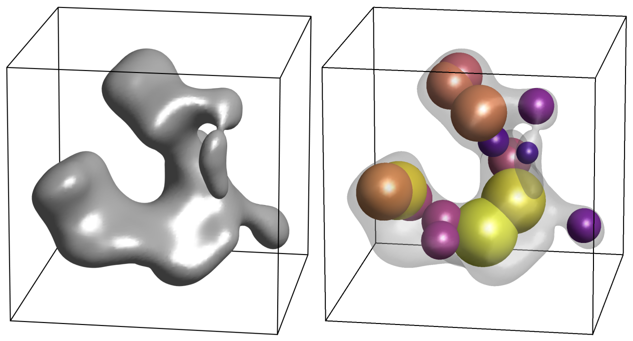
Figure 1. *Left:* An example of an object in 3D for which we can compute local thickness. *Right:* A few spheres of different radii that can be fitted inside the object depending on its local thickness. Every point inside, say, the orange sphere, has a local thickness that is equal to or greater than the radius of the sphere.

Local thickness is an often sought-after post-processing algorithm. In particular, when analyzing the microstructure of trabecular bone from volumetric data, computing the local thickness is an essential processing step. BoneJ [5], a plugin for bone image analysis in ImageJ [13], has established itself as the gold standard for local thickness computation, thanks to an accurate and efficient algorithm [6]. This is practical for the ImageJ community but difficult to incorporate into a Python-based analysis pipeline. In Python, one option is to use the PoreSpy package [7] which comes with a large number of predefined routines including an approximation of the local thickness.

Contribution We propose an algorithm for computing local thickness, which is accurate and fast. Our algorithm utilizes two approximations for more efficient computation. First, the dilation with a sphere of a large radius is approximated with consecutive dilations with a sphere of radius one. This results in a dilation not being perfectly spherical, but polygonal (polyhedral). Second (and optional), we use a scaled approach, where the local thickness is computed for a downscaled image, then upscaled, and corrected.

Our algorithm is implemented as a pip-installable Python module. The module, `localthickness`, has lightweight dependencies: apart from the NumPy module the only other dependence is `edt` module for a fast computation of Euclidean distance transform.

We tested the performance of our algorithm and compared it against BoneJ and PoreSpy algorithms in terms of accuracy and run time.

2. Related work

Computation of local thickness is central for assessing bone microstructure from X-ray microtomographic (μCT) images. Hildebrand and Rüegsegger [8] were the first to propose an algorithm for local thickness, and local thickness has since found a place in guidelines for evaluating bone microstructure [2] and numerous other applications [3, 4]. The use of local thickness has propagated to other organic structures [12], medical image analysis [14], dentistry [15], and the development of medical and dental scanners [10]. In quantitative X-ray computed tomography [16], local thickness is mentioned as a general tool and used in areas such as 3D printing [11] and advanced materials research [1].

A gold standard for computing local thickness is the algorithm [6] included in BoneJ [5]. This algorithm avoids morphological filtering of the whole object. Instead, it computes a distance ridge of the object, to identify voxels that are to be dilated. The dilation is performed by computing pairwise distances between ridge voxels and volume voxels. This leaves a small discrepancy on the surface of the object, which requires cleanup. The BoneJ algorithm gives very accurate results, and the computation is fast.

The algorithm in PoreSpy [7] computes the local thickness for a number of discrete values chosen beforehand. For each value, a dilation is computed using linear filtering followed by thresholding, where the filtering itself is done in the Fourier domain. The result of PoreSpy, as well as the run time, is strongly influenced by the number of chosen values so the user can find a compromise between accuracy and speed.

3. Conventional algorithm for local thickness

For an object in 3D, the local thickness at any point of the object is defined as the radius of the largest sphere that fits within the object and contains the point.

In other words, for an object $A \subset \mathbf{R}^3$ and a point $\mathbf{x} \in A$, the local thickness is given by

$$t(\mathbf{x}) = \max\{r \in \mathbf{R} \,:\, \mathbf{x} \in S(\mathbf{c}, r) \subseteq A,\, \mathbf{c} \in \mathbf{R}^3\}\,, \quad (1)$$

where $S(\mathbf{c}, r)$ is a closed ball centered at $\mathbf{c}$ with radius r. It is common to extend t by saying that $t(\mathbf{x}) = 0$ for $\mathbf{x}$ outside A.

In a naive implementation of this definition, calculating the local thickness for every point $\mathbf{x}$ would involve considering various points $\mathbf{c}$ and a range of radii r.

A more efficient algorithm, which we denote *conventional algorithm for local thickness*, is based on the computation of the distance field and dilation with spherical structuring elements of different radii. In this section, we show how the definition of local thickness Eq. 1 may be expressed in terms of a distance field and dilation.

A distance field is defined as

$$d(\mathbf{x}) = \max\{r \in \mathbf{R} \,:\, S(\mathbf{x}, r) \subseteq A\}\,. \quad (2)$$

Consider now a grayscale *super-level* of d

$$d_r(\mathbf{x}) = \begin{cases} d(\mathbf{x}) & r \leq d(\mathbf{x}) \\ 0 & \text{elsewhere} \end{cases} \quad (3)$$

and a grayscale dilation $d_r \oplus S_r$, where S_r is a ball with radius r. Because of the properties of dilation, distance field, super-level, and local thickness, we know that

$$(d_r \oplus S_r)(\mathbf{x}) \leq t(\mathbf{x}),\ \text{for every } r \in \mathbf{R} \quad (4)$$

and we also know that

$$\text{if } r = t(\mathbf{x}), \text{ then } (d_r \oplus S_r)(\mathbf{x}) = t(\mathbf{x})\,. \quad (5)$$

Therefore, we can conclude that

$$t = \max\{d_r \oplus S_r,\, r \in \mathbf{R}\}\,. \quad (6)$$

This formulation of local thickness is more operative than Eq. 1 as it only involves considering different values of r. In practice, this is achieved by looping over discretized radii $r \in \{1, \ldots, \text{floor}(\max(d)\}$. When using standard packages for distance field and grayscale dilation, the local thickness may be computed for all voxels in the image with just a few lines of code.

Alg. 1 is a direct implementation of Eq. 6. In the algorithm, we expect the distance transform to be computed, so the algorithm takes a distance field as input. In the pseudocode, we use square brackets to denote boolean (logical) indexing into the array, so `a[b]` denotes elements of `a` where `b` has values `True`. The intermediate super-levels are stored in the variable `df` originally used for the distance field.

Algorithm 1 Conventional local thickness

```
function CONVENTIONAL(df)
    out ← df
    for r = 1 to floor(max(df)) do
        df[df < r] ← 0
        temp ← df ⊕ s(r)
        change ← temp > out
        out[change] ← temp[change]
    return out
```

3.1. Variants of the conventional algorithm

In Alg. 1, lines 6 and 7 result with `out` being reassigned to an element-wise maximum of `out` and `temp`. Therefore, if using an element-wise maximum, the variable `change` does not need to be introduced. Furthermore, the condition `temp > out` in line 6 may be replaced with the equivalent condition `temp > 0`. Lastly, another variant of the algorithm inverts the order of dilations, such that it starts with the largest `r`. In that case, dilation needs to operate on an increasing number of elements from `df` in each iteration, so re-assignment in line 4 needs to be modified accordingly.

A lot of intermediate computation in Alg. 1 is not used in the final result, being overwritten by a larger value. The voxels that, via dilation, contribute to the final result are on the ridge of the distance field. It is therefore safe to remove all the (or some) non-ridge voxels from the distance field and apply Alg. 1. This may be advantageous if dilation is implemented such that it runs faster on sparse arrays.[1]

3.2. Time complexity of the conventional algorithm

Consider a 3D object occupying a cube as in Fig. 1. Denote the side length of the cube as X and the largest local thickness of the object as S. If the structure is imaged such that the length X is divided into x pixels, the thickness of the object in pixels is $s = \frac{S}{X}x$. The conventional computation of local thickness includes a loop

```
for r = 1 to s do
    df ⊕ s(r)
```

The computation time complexity of dilation is linear with respect to the number of voxels to be processed, in this case, x^3, and the size of the structuring element, here r^3. That is, dilation takes $\mathcal{O}(x^3r^3)$ per iteration. On the other hand, the number of iterations s grows linearly with x. Therefore, the time complexity of the loop is

$$\mathcal{O}(x^3)\sum_{r=1}^{\mathcal{O}(x)}\mathcal{O}(r^3) = \mathcal{O}(x^3)\mathcal{O}(x^4) = \mathcal{O}(x^7)\,.$$

[1]As mentioned, the local thickness algorithm in BoneJ does not dilate using structuring elements but computes pairwise distances between voxels and ridge points. Therefore, they benefit from removing non-ridge points.

Volume size x^3	512^3	1024^3	2048^3	4096^3
Conventional $\mathcal{O}(x^7)$	1 sec	2 min	4.5 h	24 days

Table 1. The expected increase in processing time for conventional local thickness, assuming that it takes 1 sec to process the smallest volume.

This leads to a rapid increase in processing time as volume size grows, see Tab. 1.

4. Fast local thickness

In the conventional algorithm, the dilation always operates on one super-level (subset) of the distance field, and the radius of the spherical structuring element is increasing in each iteration. However, by utilizing the property of the distance field, and the commutativity of dilation, it is possible to formulate an algorithm that uses the sphere of radius 1 in every iteration. To obtain the result as with the conventional algorithm, the dilation operates on *outcome* of the previous iteration. We denote this approach *fast local thickness*.

In the fast algorithm, instead of dilating with a sphere of radius n, we use n consecutive dilations with a sphere of radius 1. Furthermore, one iteration of dilation may operate on multiple super-levels simultaneously. However, we need to keep track of how many times to dilate each super-level. For this, we can use the fact that the distance between consecutive level-sets of the distance field is 1.

See Alg. 2 for one variant of this approach. The algorithm iteratively dilates the distance field with the sphere of radius 1. However, only values larger than the current iteration counter are updated. This means that a certain value of the distance field, say a value $d(x) = 5$, will freely propagate for 5 iterations, filling in the sphere of radius 5. Notice that the points in the periphery of the sphere start with a positive value (due to the property of the distance field) and are incremented by 1 in each iteration (due to the property of the distance field and the dilation) until reaching the maximal value of 5. Once the sphere is filled in, the iteration counter becomes larger than the values in the sphere, and no further update is made. In the intersections of two spheres, the values originating from the larger sphere will propagate for a larger number of iterations, correctly overwriting the smaller values.

In a continuous case, or in a 1D setting, the conventional and the fast algorithm yield the same result. However, in the discrete setting in 2D and 3D, due to the shape of the discrete 1-sphere, a dilation with a large radius cannot be achieved by many dilations with a 1-sphere.

Still, by carefully choosing the structural elements when computing dilation, we can achieve a result where consecutive dilations yield a polyhedron (polygon in 2D) well-approximating a sphere. For this, we formulate the dilation with a 1-sphere as a weighted sum of dilations with simple structuring elements.

Algorithm 2 Fast local thickness

```
function FAST(df)
    out ← df
    for r = 0 to floor(max(df))-1 do
        temp ← out ⊕ s(1)            ▷ Use DilateOne
        change ← out > r
        out[change] ← temp[change]
    return out
```

Algorithm 3 Dilate with s(1)

```
function DILATEONE(v)
    $selem_i$ = selems for 2D or 3D
    $w_i$ = weights for 2D or 3D
    out ← $\sum_i w_i$ v ⊕ $selem_i$
    return out
```

4.1. Structuring elements

The structuring elements used for our algorithm have a kernel size with a side length of 3, so we use the smallest possible kernels. In 2D, the structuring elements are a discrete disc and annuli of growing radii. In 3D, we use discrete spheres and spherical shells. We make sure that the central pixel (voxel) is always set to one. Algorithms for producing structural elements are 4 and 5. Here `s(r)` is a structuring element with ones in all pixels where the distance from the center of the pixel to the center of the kernel is smaller or equal to `r`. The resulting structuring elements can be seen in Fig. 2.

The weights w_i determine the shape of the non-binary discrete approximation of the circle. We weigh the structuring elements by the inverse of their largest radius. So the element involving $(\sqrt{3})$ is weighted proportional to $1/\sqrt{3}$. After normalizing the weights to sum to 1, we arrive at the expressions in Alg. 4 and Alg. 5. Our results show that this weighting yields a good approximation. Still, it is worth noticing that this weighing is slightly arbitrary. It turns out that weights affect the shape of the resulting approximation in a non-trivial way [9], and the quality of the approximation is difficult to assess.

How the outcome of our iterative dilation compares with dilation using a large structuring element can be seen in Fig. 3 and Fig. 4.

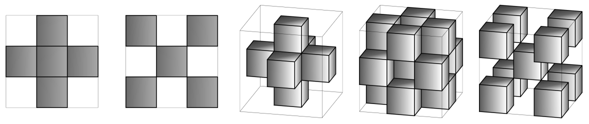

Figure 2. Structuring elements for fast local thickness in 2D and 3D.

Algorithm 4 Selems and weights for 2D

```
$selem_1$ = disk(1)
$selem_2$ = disk($\sqrt{2}$) - disk(1) + disk(0)
$w_1$ = $\frac{\sqrt{2}}{1+\sqrt{2}}$
$w_2$ = $\frac{1}{1+\sqrt{2}}$
```

Algorithm 5 Selems and weights for 3D

```
$selem_1$ = s(1)
$selem_2$ = s($\sqrt{2}$) - s(1) + s(0)
$selem_3$ = s($\sqrt{3}$) - s($\sqrt{2}$) + s(0)
$w_1$ = $\frac{\sqrt{6}}{\sqrt{6}+\sqrt{3}+\sqrt{2}}$
$w_2$ = $\frac{\sqrt{3}}{\sqrt{6}+\sqrt{3}+\sqrt{2}}$
$w_3$ = $\frac{\sqrt{2}}{\sqrt{6}+\sqrt{3}+\sqrt{2}}$
```

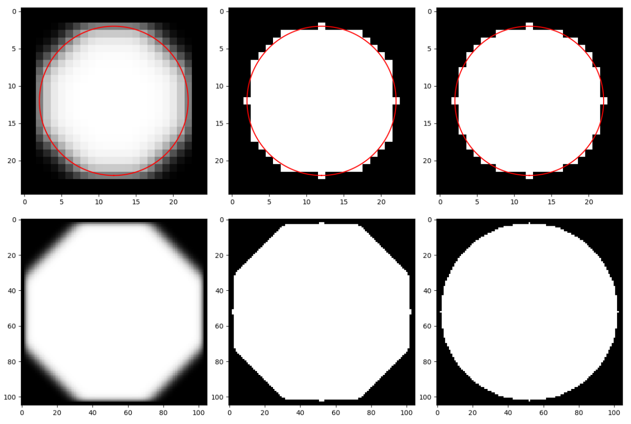

Figure 3. Dilating one white pixel with a sphere of radius 10 pixels (top line) and 50 pixels (bottom line). In the left column is the outcome of our iterative dilation. In the middle column is the iterative dilation thresholded with 0.5. The right column shows a discrete disc, for comparison. For 10 pixels we overlay a circle with radius of 10 pixels.

4.2. Variants of the fast algorithm

In Alg. 2, dilation in line 4 operates on `out`. However, it would be enough to operate only where `out` is greater than `r`, as the change may only occur there. Whether this is advantageous depends on the implementation of dilation.

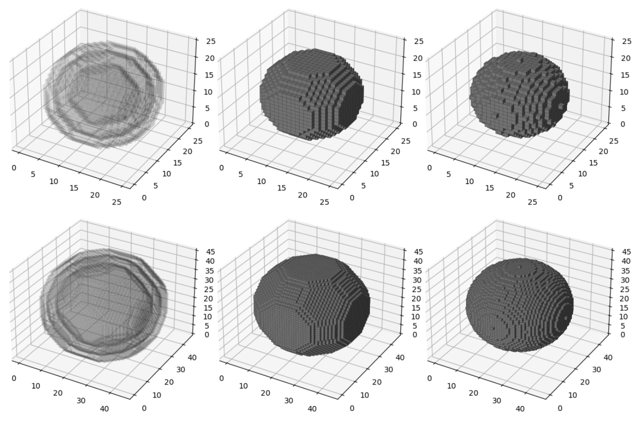
Figure 4. Dilating one white voxel with a sphere of radius 10 pixels (top line) and 20 pixels (bottom line). In the left column is the outcome of our iterative dilation, shown as isosurfaces with values 0.1, 0.5, and 0.9. In the middle column is the iterative dilation thresholded at 0.5. The right column shows a discrete ball, for comparison.

Furthermore, similar to the conventional algorithm, the fast algorithm may be formulated such that it starts with the largest `r`. Note also that the variable `df` is not used by the algorithm, so dilations may operate directly on `df`, and a new variable `out` does not need to be introduced.

4.3. Time complexity of the fast algorithm

Thanks to the constant size of the structuring element, each iteration of our algorithm has the same time complexity, that is $\mathcal{O}(x^3 1^3)$ per iteration. The number of iterations is proportional to the maximum thickness, and therefore we obtain

$$\mathcal{O}(x^3) \sum_{i=1}^{\mathcal{O}(x)} 1^3 = \mathcal{O}(x^3)\mathcal{O}(x) = \mathcal{O}(x^4)\,.$$

4.4. Scaled local thickness

In a continuous setting, shrinking the object by a factor, say 2, would result in the local thickness being reduced everywhere by the same factor. In the discrete setting, this can be utilized to speed up the computation: the volume is downscaled prior to processing, and the result is upscaled, and multiplied, after the processing. This strategy can be used with *any* of the algorithms for computation of the local thickness. The loss of accuracy when using scaling will depend on the properties of the algorithm.

Our fast algorithm is suitable for scaling. Thanks to the smooth and continuous output of the algorithm, scaling has a modest effect on the result. Furthermore, for better accuracy, we perform a distance transform before scaling, and we apply the final dilation step on the full-size volume.

4.5. Code

Python implementation of our fast local thickness can be downloaded from the GitHub repository https://github.com/vedranaa/local-thickness and the module can also be installed using `pip install localthickness`.

5. Results

We have experimentally investigated our method on 2D and 3D images. In 2D, it is possible to compute a local thickness using the conventional method for images that contain up to several hundred columns or rows. Therefore in 2D, we use the conventional method as the ground truth in our comparison. In 3D, due to the growth in computational complexity with image size, it is only possible to use the conventional method on tiny volumes. In both 2D and 3D, we compare with PoreSpy [7], and we always use 25 thickness values, which is the default value of the PoreSpy algorithm. Since PoreSpy is also a Python package, we could include a direct comparison of the run time. In 3D we also compare our method to BoneJ [5]. BoneJ is designed for 3D bone analysis and does not come with the option of computing a 2D local thickness.

The results of one 2D experiment are in Fig. 5. Without the use of scaling, the run time of our method is similar to that of PoreSpy, but in the scaled version (scale factor of 0.5), there is a significant speed-up in computation time. The histograms obtained from our methods are very similar to the ground truth, whereas the discrete nature of PoreSpy is clearly seen in the histogram. The same is seen in the cumulative histograms, where both versions of our method align with the ground truth. This is also shown in the difference maps, where our method has a slight tendency to overestimate the thickness, but has a significantly lower mean absolute difference than PoreSpy. The same is evident from the scatter plots, with points predominantly slightly below the diagonal.

In 3D, we carried out an experiment on volumes of μCT scans of human femur bones. We cropped out five samples of 250^3 voxels and made sure to cover a varying degree of cortical and trabecular bone structure. The good contrast between air and bone made it easy to binarize volumes using thresholding. In Fig. 6, we show the local thickness computed for two of these volumes. For all volumes, we computed both bone thickness and bone spacing. The thickness is computed as the local thickness of the binary volume representing the bone, and the spacing is computed as the local thickness of the inverse binary volume. Computing both the thickness and the spacing is a common practice in bone analysis.

First, we focus on run time and compare our local thickness algorithm to PoreSpy. However, we did not measure

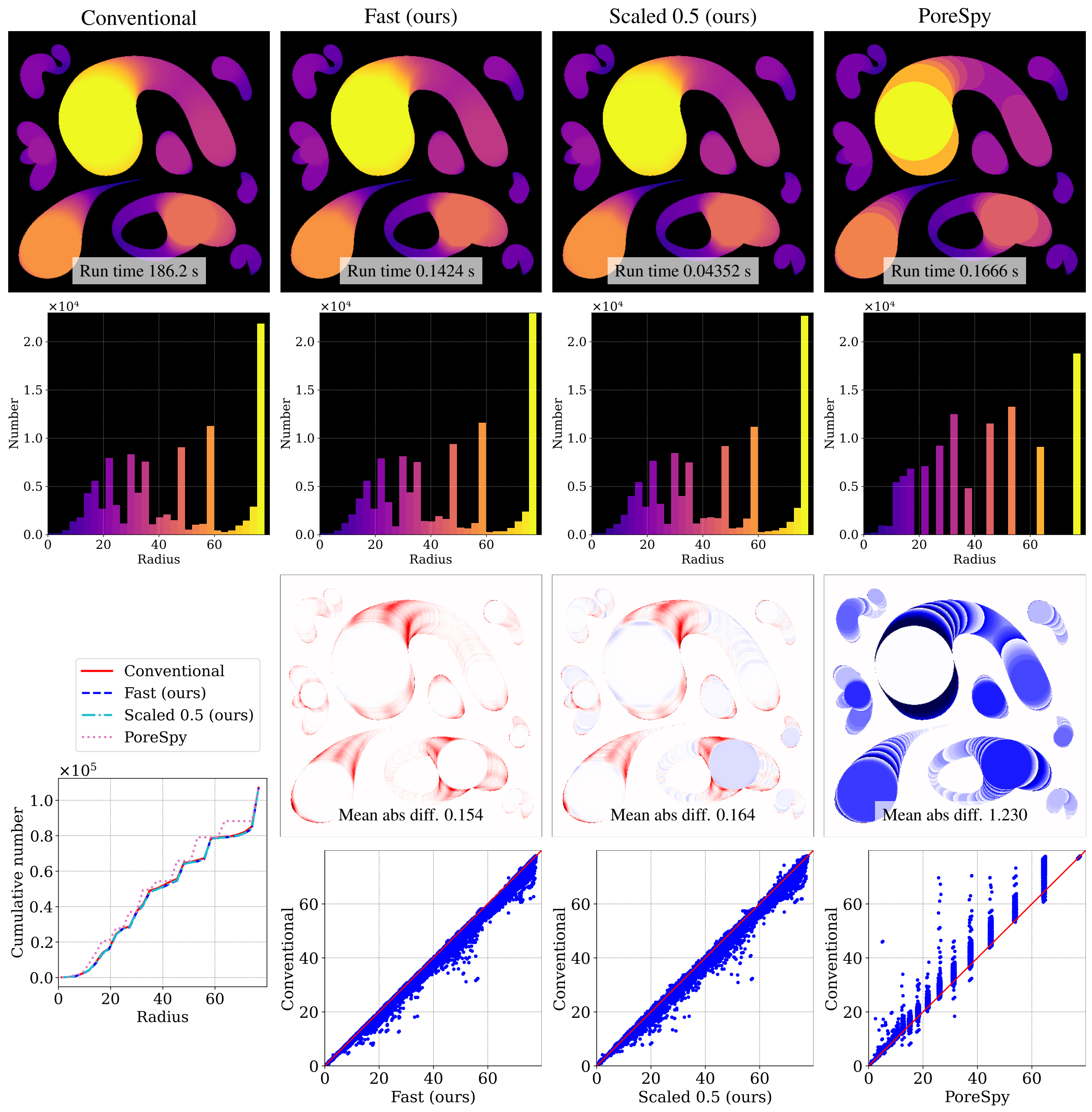

Figure 5. Qualitative and quantitative comparison of the 2D algorithms for computing the local thickness of a 512^2 image. The top row shows the local thickness computed using the conventional method (ground truth), our fast method without scaling, our method scaled with a factor of 0.5, and PoreSpy. Colormap is the same as in the thickness histograms below. Further below are the thickness images with the ground truth subtracted (red is positive, and blue negative), and a scatter plot of all the pixels in the images for a per-pixel comparison to the ground truth. The scatter plot is included to see how values differ from ground truth, but note that for our method most pixels are at, or close to, the diagonal. In the left bottom corner, we show the cumulative histogram of the four thickness computations.

the run-time for BoneJ, because it is an ImageJ [13] plugin and we could not make a precise and fair comparison. Still, computing the local thickness of a volume using BoneJ, is not faster than using PoreSpy. The measured run times are

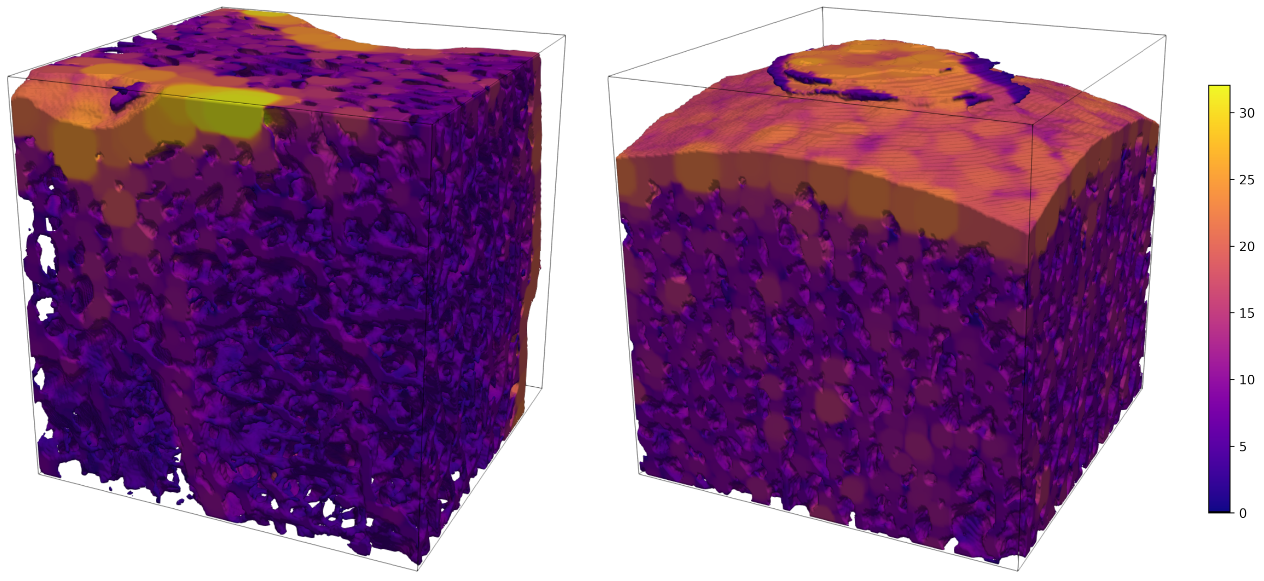

Figure 6. Local thickness of 3D volumes of bone computed using our fast local thickness approach. The volumes are of size 250^3, and the thickness is expressed in voxels that are $58.1\,\mu\text{m}$ in size.

shown in Fig. 7. When not using the scaling option, our method is in most cases slower than PoreSpy, but a significant speedup is obtained when using the scaling option. PoreSpy is almost constant in processing time because it uses a fixed number of iterations. Our method is slower in the presence of thicker structures because it needs more iterations to cover these. This is well illustrated in the five volumes we have tested, where the last volume, Vol. 5, mainly contains fine trabecular structures, and therefore our algorithm computes the local thickness very fast.

The gain in speed using the scaling option does not compromise the quality of the results obtained, which is shown in Fig. 8. Here, we have calculated the mean, standard deviation, and maximum value for bone thickness and bone spacing of the five volumes. These values are typically extracted when analyzing bone volumes and are easily extracted using the BoneJ plugin. We compare our method to BoneJ and PoreSpy. Similarly to what we saw in 2D, our method tends to compute a mean thickness that is slightly higher than BoneJ and PoreSpy, but all statistics are consistent between the three versions of our method. We always obtain the same ordering of the measures for the five volumes, and therefore our method will be reliable for measuring differences in structures, even when using a scale of 0.25. BoneJ gives results similar to our method, whereas PoreSpy is slightly less consistent. Recall that PoreSpy uses default 25 thickness values, and a better result could be obtained with a higher setting, but it would compromise its run time.

In Fig. 9, we show a cumulative histogram for one of the five analyzed volumes. There is very little difference

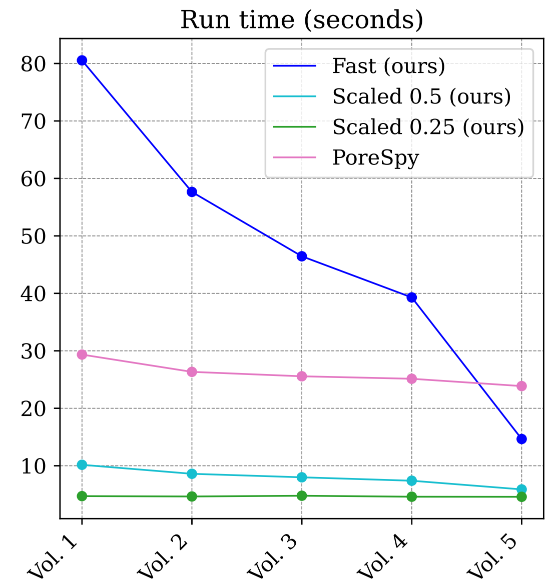

Figure 7. Run time for computing the 3D local thickness using our method and PoreSpy on five 250^3 volumes of trabecular and cortical bone taken from a larger µCT scan of a human femur. The five volumes have been chosen such that they have different-sized bone thicknesses and spacing, where Vol. 1 has relatively thick structures and Vol. 5 has predominately thin structures.

between the three versions of our method, and as also shown in 2D, PoreSpy will give a discrete thickness computation. While we do not show it in the paper, similar behavior is seen for other test volumes.

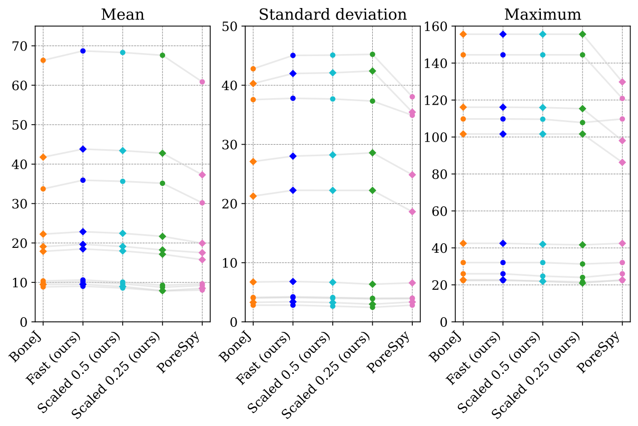

Figure 8. Statistics extracted from the bone thickness and spacing. Each line represents one of the five volumes, and values for bone thickness are shown with dots, while bone spacing is shown with diamonds. Shown are the mean, the standard deviation, and the maximum value for bone thickness and spacing.

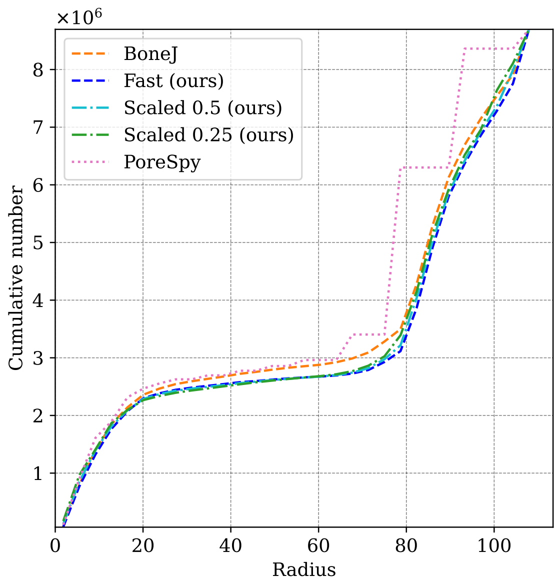

Figure 9. Cumulative histograms of the thickness computed on one of the volumes used in our experiments.

We have also experimentally established the computational complexity of our fast local thickness algorithm, as shown in Fig. 10. This experiment was carried out by computing the local thickness using our fast local thickness approach and the conventional method. We used only volumes small enough to be processed using the conventional method. When increasing the size of the volume and plotting the run time of the methods on a log-log scale, we can estimate the computational complexity as the slope of the line fitted to the results. The conventional method has a slope of 7.03, which fits very well with its theoretical complexity of $\mathcal{O}(x^7)$. Our fast local thickness obtains a slope of 3.57 which is slightly lower than the theoretical complexity of $\mathcal{O}(x^4)$.

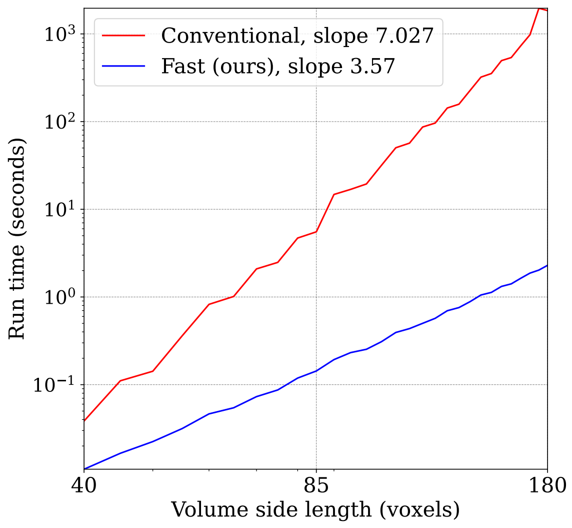

Figure 10. Computational time of fast local thickness compared to the conventional method shown in a log-log plot. This confirms that the computational complexity of the conventional method is $\mathcal{O}(x^7)$, whereas our method is $\mathcal{O}(x^4)$.

6. Conclusion

We contribute with a new method to compute the local thickness of structures in 2D or 3D, which is faster than existing alternatives and gives similar or better results. Our method utilizes the fact that local thickness can be computed by iteratively dilating a distance field using a few small structuring elements. The advantage of our approach is that the computed thickness is smooth, and gives a thickness histogram that is very close to the ground truth obtained using the conventional method. The computational complexity of our method is $\mathcal{O}(x^4)$ compared to $\mathcal{O}(x^7)$ for the conventional method. We have implemented our method in Python and made it freely available as a pip-installable module with minimal dependencies. This is a good and simple alternative to existing methods and makes it easy for the microscopy community to access an important tool for the quantitative analysis of 3D structures.

7. Acknowledgements

This work was supported by a research grant (VIL50425) from VILLUM FONDEN and the infrastructure for Quantitative AI-based Tomography (QUAITOM) supported by a Novo Nordisk Foundation Data Science Programme (grant number NNF21OC0069766).

References

[1] Hyosung An, John W Smith, Bingqiang Ji, Stephen Cotty, Shan Zhou, Lehan Yao, Falon C Kalutantirige, Wenxiang Chen, Zihao Ou, Xiao Su, et al. Mechanism and performance relevance of nanomorphogenesis in polyamide films revealed by quantitative 3D imaging and machine learning. *Science Advances*, 8(8):eabk1888, 2022. 2

[2] Mary L Bouxsein, Stephen K Boyd, Blaine A Christiansen, Robert E Guldberg, Karl J Jepsen, and Ralph Müller. Guidelines for assessment of bone microstructure in rodents using micro–computed tomography. *Journal of bone and mineral research*, 25(7):1468–1486, 2010. 2

[3] Helen R Buie, Graeme M Campbell, R Joshua Klinck, Joshua A MacNeil, and Steven K Boyd. Automatic segmentation of cortical and trabecular compartments based on a dual threshold technique for in vivo micro-CT bone analysis. *Bone*, 41(4):505–515, 2007. 2

[4] David ML Cooper, Andrei L Turinsky, Christoph Wilhelm Sensen, and Benedikt Hallgrímsson. Quantitative 3D analysis of the canal network in cortical bone by micro-computed tomography. *The Anatomical Record Part B: The New Anatomist: An Official Publication of the American Association of Anatomists*, 274(1):169–179, 2003. 2

[5] Michael Doube, Michał M Kłosowski, Ignacio Arganda-Carreras, Fabrice P Cordelières, Robert P Dougherty, Jonathan S Jackson, Benjamin Schmid, John R Hutchinson, and Sandra J Shefelbine. BoneJ: free and extensible bone image analysis in ImageJ. *Bone*, 47(6):1076–1079, 2010. 1, 2, 5

[6] Robert Dougherty and Karl-Heinz Kunzelmann. Computing local thickness of 3D structures with ImageJ. *Microscopy and Microanalysis*, 13(S02):1678–1679, 2007. 1, 2

[7] Jeff T Gostick, Zohaib A Khan, Thomas G Tranter, Matthew DR Kok, Mehrez Agnaou, Mohammadamin Sadeghi, and Rhodri Jervis. PoreSpy: A python toolkit for quantitative analysis of porous media images. *Journal of Open Source Software*, 4(37):1296, 2019. 1, 2, 5

[8] Tor Hildebrand and Peter Rüegsegger. A new method for the model-independent assessment of thickness in three-dimensional images. *Journal of microscopy*, 185(1):67–75, 1997. 2

[9] Patrick M Jensen, Camilla H Trinderup, Anders B Dahl, and Vedrana A Dahl. Zonohedral approximation of spherical structuring element for volumetric morphology. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*, pages 128–139. Springer, 2019. 4

[10] Delphine Maret, Norbert Telmon, Ove A Peters, B Lepage, J Treil, JM Inglèse, A Peyre, Jean-Luc Kahn, and Michel Sixou. Effect of voxel size on the accuracy of 3D reconstructions with cone beam CT. *Dentomaxillofacial Radiology*, 41(8):649–655, 2012. 2

[11] David G Moore, Lorenzo Barbera, Kunal Masania, and André R Studart. Three-dimensional printing of multicomponent glasses using phase-separating resins. *Nature materials*, 19(2):212–217, 2020. 2

[12] D Müter, HO Sørensen, Jette Oddershede, KN Dalby, and SLS Stipp. Microstructure and micromechanics of the heart urchin test from X-ray tomography. *Acta Biomaterialia*, 23:21–26, 2015. 2

[13] WS Rasband. ImageJ: Image processing and analysis in Java. *Astrophysics Source Code Library*, pages ascl–1206, 2012. 1, 6

[14] Punam K Saha, Robin Strand, and Gunilla Borgefors. Digital topology and geometry in medical imaging: a survey. *IEEE Transactions on Medical Imaging*, 34(9):1940–1964, 2015. 2

[15] Naoya Taniguchi, Shunsuke Fujibayashi, Mitsuru Takemoto, Kiyoyuki Sasaki, Bungo Otsuki, Takashi Nakamura, Tomiharu Matsushita, Tadashi Kokubo, and Shuichi Matsuda. Effect of pore size on bone ingrowth into porous titanium implants fabricated by additive manufacturing: an in vivo experiment. *Materials Science and Engineering: C*, 59:690–701, 2016. 2

[16] Philip J Withers, Charles Bouman, Simone Carmignato, Veerle Cnudde, David Grimaldi, Charlotte K Hagen, Eric Maire, Marena Manley, Anton Du Plessis, and Stuart R Stock. X-ray computed tomography. *Nature Reviews Methods Primers*, 1(1):18, 2021. 2