Fast local thickness

Vedrana Andersen Dahl and Anders Bjorholm Dahl

***Abstract*—We propose a fast algorithm for computation of local thickness in 3D and 2D. Our fast algorithm computer local thickness very similar to conventional algorithm, in just a fraction of time. Our algorithm is implemented in python and is freely available as a pip-installable module.**

I. Background

Contemporary high-resolution 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 using a kernel which also grows cubically in size; And processing large images often requires using large kernels.

For this reason, efficient 3D analysis requires re-visiting processing algorithms. In this paper, we consider the computation of local thickness. As we show, t ime complexity of 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.

II. Introduction

For an object in 3D, the local thickness in any point of the object is defined as the radius of the larges 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.

While there are a few efficient alternatives to the conventional approach [4][5], we propose an algorithm which is even faster, and only slightly reduces the quality of computed local thickness.

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

III. Related Work

BoneJ [4] simplifies the search set to the distance ridge. This leaves small discrepancy at the surface, which requires cleanup. The result is practically identical to the result of the conventional algorithm.

Porespy [5] use a range of dilate values, and morphological operations are implemented to utilize fft-based filtering. The result is less smooth, but comparable to the result of the conventional algorithm.

VAND: Other related papers: [1] 3700 citations, [2] 589 citations, [3] 245 citations, [7] 181 citations, [6] 1895 citations, [8] 11 citations, [9] 85 citations, [10] 583 citations.

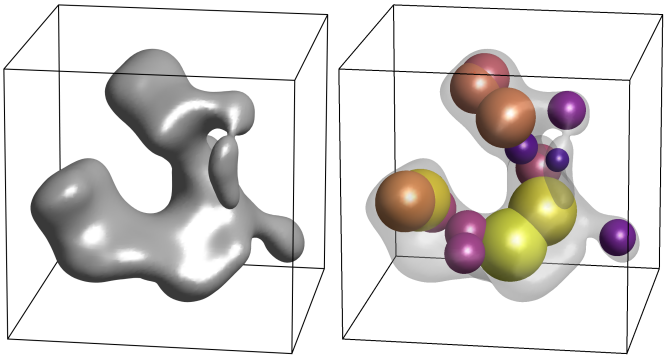


Fig. 1. *Left:* An example of an object in 3D which has local thickness. *Right:* A few spheres of different radius which can be fitted inside the object, depending on its local thickness. Every point inside, say, the orange sphere has the local thickness which is equal or larger than the radius of the sphere.

IV. Conventional algorithm for local thickness

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

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

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

where $S(\mathbf{c}, r)$ is a closed ball 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, the computation of 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 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 distance field and dilation.

A distance field is defined as

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

Consider a grayscale *super-level* of d

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

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

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

and we also know that

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

So we can conclude that

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

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 direct implementation of Eq. 6. In the algorithm, we expect the distance transform to be computed, so algorithm takes 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 superlevels are stored in the variable originally used for distance field.

Algorithm 1 Conventional local thickness

```
1: function CONVENTIONAL(df)
2:     out ← df
3:     for r = 1 to floor(max(df)) do
4:         df[df < r] ← 0
5:         temp ← df ⊕ s(r)
6:         change ← temp > out
7:         out[change] ← temp[change]
8:     return out
```

A. 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 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 points 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 points 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]

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

TABLE I
THE EXPECTED INCREASE IN PROCESSING TIME FOR CONVENTIONAL LOCAL THICKNESS, ASSUMING THAT IT TAKES 1 SEC TO PROCESS THE SMALLEST VOLUME.

B. Time complexity of the conventional algorithm

Consider a 3D object occupying a cube as in Fig. 1. Denote the 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 length X is divided in 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)
```

Computation time complexity of dilation is linear in 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 $\mathcal{O}(x^3 r^3)$ per iteration. On the other hand, the number of iterations s grows linearly with x. The time complexity of the loop is therefore

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

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

V. 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 algorithms which use the sphere of radius 1 in every iteration. To obtain the result as with the conventional algorithm, the dilation operates on the *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 distance field with the sphere

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

of radius 1. However, only values larger of 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 distance field), and are incremented by 1 in each iteration (due to the property of distance field and the dilation) until reaching the maximal value of 5. Once the sphere is filled in, the iteration counter becomes larger that the values in the sphere, and no further update is going to be made. In the intersections of two spheres, the values originating from the larger sphere will will propagate for a larger number of iterations, correctly overwritting the smaller values.

In a continuous case, or in a 1D setting, (VAND: Add figure with 1D example. Maybe also continous-like case.) the conventional and the fast algorithm yield the same result. However, in 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) 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

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

Algorithm 3 Dilate with s(1)

```
1: function DILATEONE(v)
2:     selem_i = selems for 2D or 3D
3:     w_i = weights for 2D or 3D
4:     out ← Σ_i w_i v ⊕ selem_i
5:     return out
```

A. Structuring elements

The structuring elements used for our algorithm have kernel size length of 3, so we use 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 distance from pixel center to kernel center is smaller or equal to `r`. The resulting structuring elements may be seen in the Fig. 2.

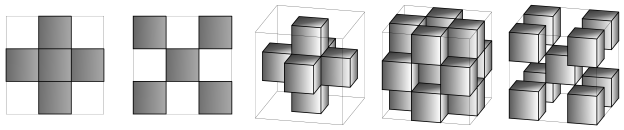


Fig. 2. Structuring elements for fast local thickness in 2D and 3D.

The weights w_i determine the shape of the non-binary discrete approximation of the circle. We weight 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 to 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 nontrivial way, and the quality of the approximation is difficult to assess.

How the oucome of our iterative dilation compares to dilation using a large sturctural element can be seen in Fig. 3 and Fig. 4.

Algorithm 4 Selems and weights for 2D

```
1: selem_1 = disk(1)
2: selem_2 = disk(√2) - disk(1) + disk(0)
3: w_1 = √2/(1+√2)
4: w_2 = 1/(1+√2)
```

Algorithm 5 Selems and weights for 3D

```
1: selem_1 = s(1)
2: selem_2 = s(√2) - s(1) + s(0)
3: selem_3 = s(√3) - s(√2) + s(0)
4: w_1 = √6/(√6+√3+√2)
5: w_2 = √3/(√6+√3+√2)
6: w_3 = √2/(√6+√3+√2)
```

B. Variants of the fast algorithm

In Alg. 2 dilation in line 4 operates on `out`. It would however be enough to operate only where `out` is larger than `r`, as only there change may occur. Whether this is advantageous depends on the implementation of dilation. Furthermore, similar to the conventional algorithm the fast algorithm may be formulated such that it starts with the largest `r`. Notice 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.

C. Time complexity of the fast algorithm

The computational complexity of our method is constant for the size of the structuring element 1^3, that is $\mathcal{O}(x^3 1^3)$ per iteration, therefore we obtain

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

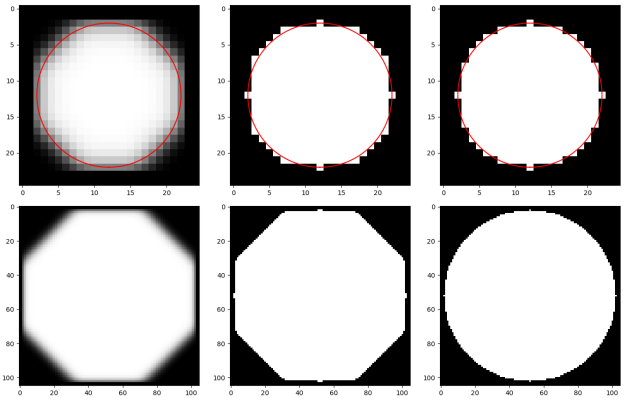


Fig. 3. Dilating one white pixel with sphere of radius 10 pixels (top line) and 50 pixels (bottom line). In the left column is the outcome of itarative dilation. In the middle column is the iterative dilation thresholded with 0.5. The right column shows a discrete disc, for comparison. For 10 pixels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

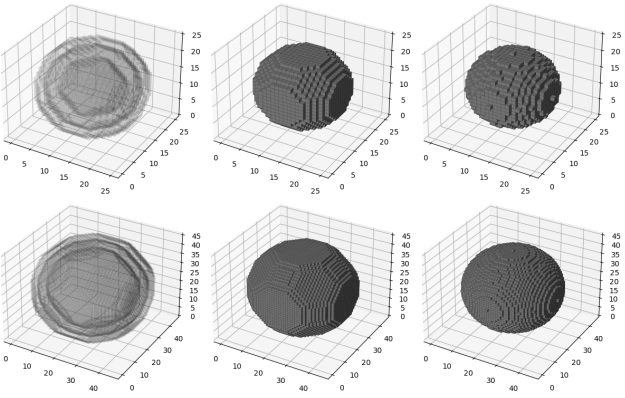


Fig. 4. Dilating one white voxel with sphere of radius 10 pixels (top line) and 20 pixels (bottom line). In the left column is the outcome of itarative dilation, shown as isosurfaces with values 0.1, 0.5 and 0.9. In the middle column is iterative dilation thresholded, but only the isosurface for value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

D. *Local thickness with scaling*

In continuous setting, shrinking the object with a factor, say 2, would result in local thickness being reduced everywhere by the same factor.

This can be utilized to speed up the computation, with any algorithm. VAND: Describe.

We add cleanup.

E. *Code*

Python code is available in the github repository https://github.com/vedranaa/local-thickness and the module can also be installed using `pip install localthickness`.

VI. Results

See Fig. 5 and notebooks. Runtime Fig. VI.

VAND: Copy results from the notebooks: 2D notebook 3D notebook

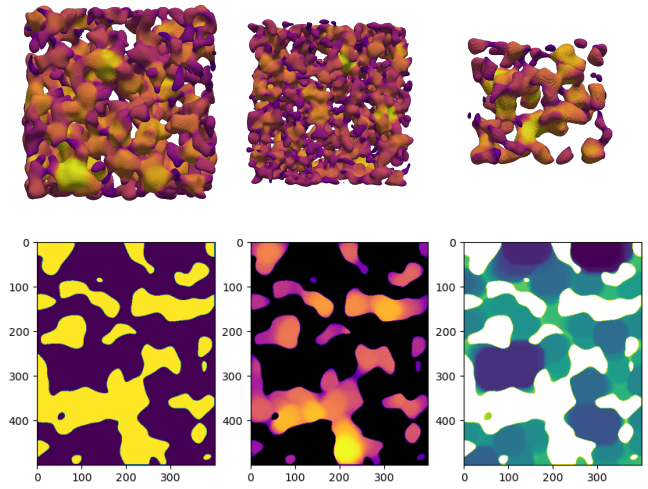
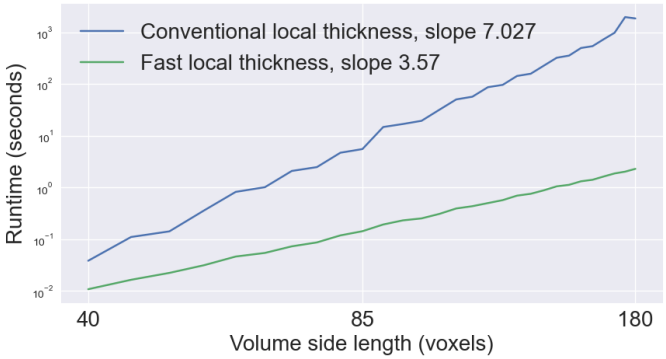


Fig. 5. A placeholder for some results.



VII. Conclusion

VAND: The conclusion goes here. May also be copied from the notebooks.

References

[1] 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] 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.

[3] 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.

[4] Robert Dougherty and Karl-Heinz Kunzelmann. Computing local thickness of 3d structures with imagej. *Microscopy and Microanalysis*, 13(S02):1678–1679, 2007.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.