Slice, Simplify and Stitch: Topology-Preserving Simplification Scheme for Massive Voxel Data

Hubert Wagner
University of Florida, Gainesville, Fl, USA

Abstract

We focus on efficient computations of topological descriptors for voxel data. This type of data includes 2D greyscale images, 3D medical scans, but also higher-dimensional scalar fields arising from physical simulations. In recent years we have seen an increase in applications of topological methods for such data. However, computational issues remain an obstacle.

We therefore propose a streaming scheme which simplifies large 3-dimensional voxel data – while provably retaining its persistent homology. We combine this scheme with an efficient boundary matrix reduction implementation, obtaining an end-to-end tool for persistent homology of large data. Computational experiments show its state-of-the-art performance. In particular, we are now able to robustly handle complex datasets with several billions voxels on a regular laptop.

A software implementation called Cubicle is available as open-source: `https://bitbucket.org/hubwag/cubicle`.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Computational topology, topological data analysis, topological image analysis, persistent homology, persistence diagram, discrete Morse theory, algorithm engineering, implementation, voxel data, volume data, image data

Digital Object Identifier 10.4230/LIPIcs.SoCG.2023.60

Supplementary Material *Software*: `https://bitbucket.org/hubwag/cubicle`
archived at `swh:1:dir:5f25601ea576ea1a004c37d94e2e2f5b94b9c00d`

Funding Supported by the 2022 Google Research Scholar Award in Algorithms and Optimization.

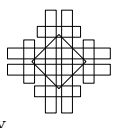
Acknowledgements I would like to thank Herbert Edelsbrunner, Teresa Heiss, Kevin Knudson, Marian Mrozek, Georg Osang and Vanessa Robins for their helpful comments.

1 Introduction

Persistent homology is one of the most popular tools offered by the field of Topological Data Analysis (TDA). It provides a rich geometric-topological descriptor of data called the persistence diagram. Point cloud data is a natural choice and a significant portion of theory, and algorithms and software focuses on this setting.

However, persistent homology is also becoming increasingly useful for other types of data – especially when used in conjunction with modern machine learning tools [23, 24, 37]. In this paper, we turn our interest to scalar voxel data in dimension 2 and 3. This type of data includes 2D gray-scale images, 3D medical scans, but also scalar fields coming from physical simulations. Intermediate results of convolutional neural networks (i.e. feature maps) are another interesting case. In all these settings, voxel data encodes potentially useful – and often intricate – geometry and topology.

One key challenge is that such datasets are often large – counted in billions of voxels or more. Currently, for such data we can compute topological descriptors such as connected components [29], merge trees [32], contour trees [17] and the Euler characteristic curve [21, 38]. While these are useful tools, there is a compromise: the first three discard higher-order topological information, and the last one forfeits information about the persistence of topological features.



Our goal is therefore to provide a method for exact computation of persistent homology of large 3D voxel data. We propose an efficient streaming computation scheme, prove its correctness and benchmark its implementation[1]. In particular, we experimentally show that for large data it is the fastest method available. Our method also uses significantly less memory, which is crucial because memory usage is the main bottleneck of existing methods. With these improvements we were able to apply our method to several practical datasets with up to 2048^3 voxels on a regular laptop. Existing software was limited to 256^3 or 512^3.

On a technical side, the efficiency of our approach is achieved by (1) *streaming* the input slice by slice, (2) efficient parallel implementation and (3) realizing selected parts of the pipeline as *external-memory* algorithms. This allows us to save memory, while ensuring that topological features spanning multiple slices are correctly captured.

Focus of the paper. We focus on: (1) conveying the main ideas behind our computation scheme; (2) contrasting it with existing approaches; (3) experimentally comparing our implementation with existing ones using practical datasets. We also discuss the most important algorithmic and implementational decisions.

Method preview. Input is a 3-dimensional array of scalar values. It is often called a **volume** and is composed of 3-dimensional **voxels**. Output is the persistence diagram of the input volume, based on an simplified intermediate representation.

> Our scheme cuts the input volume into *slices*. A small number of slices is streamed from disk and processed in parallel. Each slice is *simplified* independently using discrete Morse theory, and boundary information of this smaller representation is output to disk. Carefully handling the *border* between adjacent slices allows us to *stitch* this local information back together. In practice, we do this by constructing a global boundary matrix using the partial information coming from each slice. Despite containing extra stitching information, the resulting boundary matrix is much smaller than the boundary matrix of the input. Finally, we retrieve the persistence diagram by running a specialized version of Gaussian elimination on the resulting matrix.

2 Standard background

In this section we cover the usual theoretical background relevant for topology of voxel data. This includes cubical complexes, discrete Morse theory and persistent homology. Whenever possible we choose simple definitions that help map concepts to computations. In the next section we cover some additional definitions specific to our approach.

Cubical cells and complexes. Following [26] we define a degenerate interval $[k, k] \subset \mathbb{R}$, and a regular interval $[k, k+1] \subset \mathbb{R}$ for a natural number k. Taking a product of D intervals, p of which are regular, gives us a p-dimensional **cubical cell** in embedding dimension D. We call them cubical p-cells, or simply cells; $dim(\sigma)$ gives the dimension of cell σ. For $p = 0, 1, 2, 3$, we talk about vertices, edges, squares and cubes. Cell σ is a **face** (**coface**) of another cell τ whenever $\sigma \subset \tau$ ($\tau \subset \sigma$); it is a **proper** face or coface if additionally their dimensions differ by one. A D-dimensional cell is called the top-dimensional cell which we identify with a **voxel**. A **cubical complex**, $\mathbf{K}$, of dimension D is a finite collection of cells of dimension at

[1] Software is available as open source: `https://bitbucket.org/hubwag/cubicle/`

most D that is closed under taking faces. Namely, $\sigma \subset \tau$ implies $\sigma \in \mathbf{K}$. Given a cubical complex with n cells indexed from 1 to n, its **boundary matrix** is an $n \times n$ binary matrix M; $M[i,j] = 1$ if and only if the j-th cell is a face of the i-th cell in the complex. The columns of M encode the **boundary** of each cell.

Filtered cubical complexes. We assume a common convention in which the values assigned to input voxels are interpreted as the values of the top-dimensional cells of a cubical complex. These values are then extended to the lower dimensional faces: each cell inherits the minimum value of its top-dimensional cofaces: $val(\sigma) = min\{val(\tau) : \sigma \subset \tau \text{ and } dim(\tau) = D\}$. For us a **filtered cubical complex** is simply a cubical complex with cell values assigned as above. With this we talk about the **sublevel complex** at a value threshold t. Denoted $\mathbf{K}_{\leq t}$, it contains all cells in $\mathbf{K}$ with value not exceeding t. Clearly $\mathbf{K}_{\leq s} \subseteq \mathbf{K}_{\leq t}$ whenever $s \leq t$, which lets us define a **filtration** of cubical complexes: $\emptyset = \mathbf{K}_0 \subset \mathbf{K}_1 \cdots \subset \mathbf{K}_n = \mathbf{K}$.

Cubical homology and persistent homology. Chains of cubical cells, as well as chain, cycle, boundary, homology and persistent homology groups can be defined [33] in complete analogy to the standard simplicial case [13]. We will work with homology and persistent homology with $\mathbb{Z}_2$ coefficients, which is standard in applications.

Instead of defining these concepts formally, we offer a brief intuitive overview, which may be more useful. Intuitively, for 3-dimensional cubical complexes, homology groups capture the connected components, 1-dimensional closed loops, and voids made of cells. We call them **homological features** of dimension $0, 1, 2$ respectively, as shown in Figure 1.

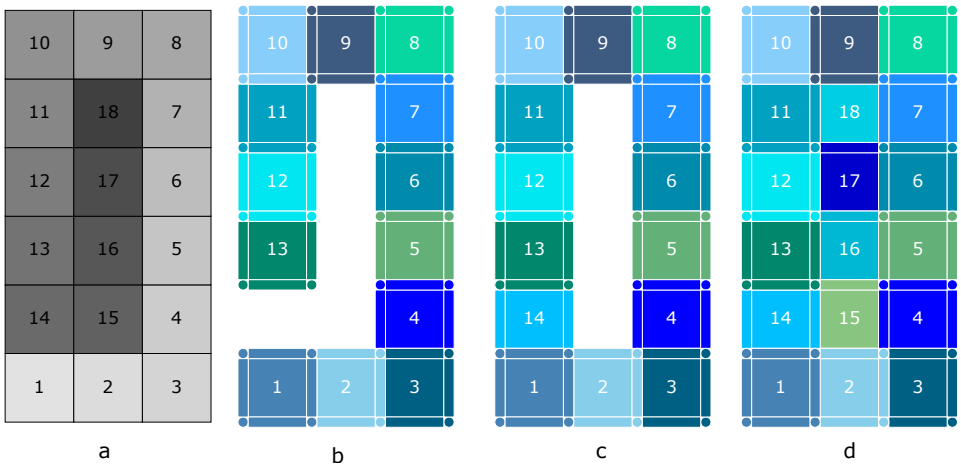


Figure 1 (a) Input volume. (b)–(d) three subcomplexes of the corresponding filtered cubical complex subdivided into blocks. At value 14 a 1-dimensional cycle is formed and it is filled in at value 18, forming a persistence pair (14,18) with persistence 4.

Persistent homology works with filtered complexes and assumes a dynamic view of data: we add voxels one by one ordered by non-decreasing values. Each voxel introduces all its faces – unless they were already present. Now, theory tells us that each p-cell can either create a p-dimensional homological feature, or destroy a $(p-1)$-dimensional one [13]. We keep track of the values of birth and death of each feature. Each such pair is called a **persistence pair**, and **persistence** itself is the lifetime of each feature. A multiset of persistence pairs forms a **persistence diagram** also called a persistence barcode. It turns out to be a powerful topological descriptor, largely due to its stability properties [14].

A **filtered boundary matrix** is the boundary matrix of a filtered cubical complex such that the rows and columns are sorted by the value of the corresponding cells. Persistent homology can be efficiently computed using Gaussian elimination of this matrix [13, 4]. The main shortcoming of this approach is the large size of this boundary matrix arising from the original input, even if stored in sparse format.

Discrete Morse theory. Since cubical complexes can be viewed as CW-complexes, Forman's discrete Morse theory [15] – **DMT** for short – can be used in this setting. We emphasize that restricting the setting to cubical complexes is crucial for computational efficiency reasons. In short, we use DMT to simplify the input while retaining crucial topological information.

DMT relies on a **discrete Morse matching** which pairs cells in a certain way; it is often called a discrete Morse vector field or discrete Morse gradient. We represent it by a directed graph, which we call the **matching graph**. Its nodes correspond to cells in the cubical complex and initially each directed edge corresponds to proper face relation between two cells. Pairing two cells corresponds to flipping an edge in this graph. A matching is a **valid** discrete Morse matching if: (1) each node in the corresponding matching graph is incident to at most one flipped edge, and (2) the corresponding matching graph is acyclic. A cell is called **critical** if it is not paired with any other cell. We define an **alternating path** between a p-cell σ and $(p-1)$-cell τ as any path in the modified graph that starts at σ ends at τ and alternates between cells in dimension p and $p-1$. They are also called V-paths, discrete Morse flow lines and gradient paths.

A **discrete Morse complex** arising from a matching is generated by its critical cells; the boundary relation between elements σ and τ is 1 if and only if there is an **odd** number of alternating paths between them, and 0 otherwise. As before, we encode this information in a boundary matrix. We remark that the parity criterion comes from using homology with $\mathbb{Z}_2$ coefficients – in more general setups this is slightly more complicated [31]. The key consequence of DMT is that the discrete Morse complex of a given complex yields homology groups which are isomorphic to the homology groups of the original complex.

In the context of filtrations, we restrict ourselves to pairing cells with the same filtration value. This process yields **filtered discrete Morse complexes** in which the filtration values are inherited from the critical cubical cells. As before, we encode it as a filtered boundary matrix. We also view it as a filtration of discrete Morse complexes: $\mathbb{M}(\mathbf{K_0}) \subset \mathbb{M}(\mathbf{K_1}) \subset \cdots \subset \mathbb{M}(\mathbf{K_n}) = \mathbb{M}(\mathbf{K})$.

3 Setup: blocks, slices and borders

In this section we define a number of non-standard definitions which are crucial for our approach. First, we define the **extended value** of a voxel as the pair (input value, index of the voxel in the input volume). These pairs are compared lexicographically. This is a technicality which breaks ties between voxels of equal value. The **block** of a voxel is the subset of its faces which share its extended value. It is useful to imagine that voxels appear one by one starting with the lowest value. Each voxel introduces all its faces which were not yet present in the complex – namely the block. We remark that an individual block is not a cubical complex – just a subset of cubes. However, the blocks of all voxels in a cubical complex disjointly decompose this complex. See Figure 1 for an illustration.

We cut the input volume – and its cubical complex – in the following way. We imagine a horizontal hyperplane at integer height h intersecting the cubical complex. The cells contained in this plane form the **border** which is also the intersection of the two resulting **slices**. This border forms a cubical complex with the dimension of the highest dimensional cell equal to $D-1$. If we cut k times, we generally get $k+1$ slices separated by k borders.

We define the **interior** of each slice as the slice itself minus its borders. By a **stratum** we mean either the interior or a border of a slice. The complex disjointly decomposes into strata. Later we will compute discrete Morse matchings separately for each stratum, noting that internal strata are not cubical complexes, but all border strata are.

We remark that in actual computations we will cut the input volume into overlapping slices. More precisely, with each slice we load an extra layer of voxels belonging to each adjacent slice. Each extra layer has height one and is called an **overlap**. It will allow us to assign values to cells belonging to the border strata consistently across adjacent slices.

4 Related work

In this section we overview of existing literature in the topic. We focus on work related to persistent homology and discrete Morse complexes in the context of voxel data. In particular, we emphasize existing techniques which we use in our current approach.

To the best of our knowledge, applying algebraic topology in the context of voxel data were pioneered in the form of shape functions by Verri, Uras, Frosini and Ferri [35]. Using higher-degree homological tools go back to the work of Kaczyński, Mischaikow and Mrozek in the context of dynamical systems [30], with extensions to other application domains [26].

The early 2010s saw an increased interest in computing topological descriptors for voxel data. Bendich, Edelsbrunner and Kerber proposed an efficient approximation scheme for persistent homology of 3D voxel data [6]. In contrast, our approach aims at exact computations. An efficient approach for exact computation of persistence of voxel data of arbitrary dimension is due to Wagner, Chen and Vucini [36]. It relies on efficient generation of the boundary matrix of the entire complex. We reuse some of the techniques used in this paper, in particular the data-structure for compact storage of information for each cell. An efficient implementation of this approach is provided in Gudhi [12] and DIPHA [3]. One downside of this scheme is the large size of the boundary matrix – which prompted development of simplification methods, like the one below.

Work by Robins, Wood, Sheppard [33] marks a breakthrough in computing topological descriptors for voxel data. We refer to this work as $\mathcal{RWS}$ and describe it in more detail in Section 5. In short, it preprocesses the input using discrete Morse theory typically resulting in a much smaller boundary matrix encoding the same topology. In [19], certain algorithmic aspects were improved and a memory-efficient storage of the matching was proposed. This allowed for handling large data – but significantly complicated the implementation. Since our new approach needs to store only a single slice of data at a time, we use a simpler but equally efficient implementation without affecting memory usage.

Work by Mischaikow and Nanda [31] generalizes the theory and algorithms presented in $\mathcal{RWS}$ beyond cubical complexes. A related software package, Perseus, has been a popular tool capable of computing persistence for a variety of input types. One lesson learned from this work was the trade-off between flexibility and efficiency. For voxel data the preprocessing step turned out to be more costly than computing persistence directly [4]. With that in mind, our approach specializes in voxel data. This paper also reports the first approach for streaming preprocessing – each levelset was stored separately, simplified and merged together later. In contrast, we use a spatial decomposition of the volume.

Discrete Morse theory was also used to perform more aggressive simplification. This proved to be useful in data visualization. One particularly impressive approach is due to Gyulassy, Bremer, Hamann and Pascucci [20]. Already in 2008 it handled a volume with a billion voxels on commodity hardware in a day. However, the simplified complex computed

in this approach cannot be used to compute persistent homology of the original data [9]. Still, our approach is inspired by the data subdivision scheme used in this approach. Despite certain similarities, the details of our subdivision scheme, subsequent computations and final goal are all different. Our method can be viewed in terms of discrete stratified Morse theory of Knudson and Wang [28].

Concurrently to the development of DMT-based methods, matrix reduction algorithms were significantly improved [7, 2, 4, 22, 3]. In Section 7 we describe how we adapted certain techniques from the PHAT library [4] at the preprocessing level.

Bauer's Ripser [1] was originally devised for the Vietoris–Rips construction arising from point cloud data. Recently Ripser was adapted to voxel data with CubicalRipser [27].

Finally, we remark that the proposed scheme was first conceived in 2015 and preliminary results were informally presented at two international meetings at 2017. Software development began in 2015, and the first stable version of the software was published in 2018.

5 Persistence-aware simplification scheme

In this section, we overview an existing approach by by Robins, Wood and Sheppard [33], ($\mathcal{RWS}$), for simplifying cubical complexes stored in memory.

The goal is to compute a smaller representation of data that encodes the same persistent homology as the input volume. More precisely output persistence diagrams are allowed to differ only by **ephemeral persistence pairs** – each corresponding to a feature that is born and dies at the same filtration value. Such pairs are discarded in practice anyway. The crucial observation by Robins and collaborators is that ephemeral features abound in cubical filtrations, allowing them to propose an efficient simplification scheme.

This method computes a valid discrete Morse matchings *separately* for each block. It then computes the boundary matrix of a filtered discrete Morse complex which encodes the same persistent homology. This step is done by computing the parity of the numbers of alternating paths between pairs of critical cells. See Figure 2 for illustration. We discuss correctness of this approach for filtered cubical complexes.

Validity of the matching. We show that if the matching graph within each block is acyclic, then the global one is acyclic as well. We observe that the *extended* value is generally non-increasing along cells in an alternating path. However leaving a block necessarily decreases the extended value. Since forming a cycle would require leaving a block and returning, cycles cannot form since the extended values along paths cannot increase.

Topologial correctness of simplification. Applying DMT on each block yields a complex that encodes the same persistent homology – up to ephemeral pairs. Indeed, this block-wise construction yields the filtration: $\mathbb{M}(\mathbf{K_0}) \subset \mathbb{M}(\mathbf{K_1}) \subset \ldots \mathbb{M}(\mathbf{K_n}) = \mathbb{M}(\mathbf{K})$. Now, Forman's theory tells us that $H_*(\mathbb{M}(\mathbf{K_i})) = H_*(\mathbf{K_i})$. Applying the Persistence Equivalence Theorem [13] yields the desired result.

Hardness of matchings. Generally, finding discrete Morse matchings minimizing the number of critical cells is a computationally hard problem [25]. However, $\mathcal{RWS}$ showed a simple $\Theta(b \log b)$ time algorithm for optimal matching for a block of a 3-dimensional cubical complex with b cells.

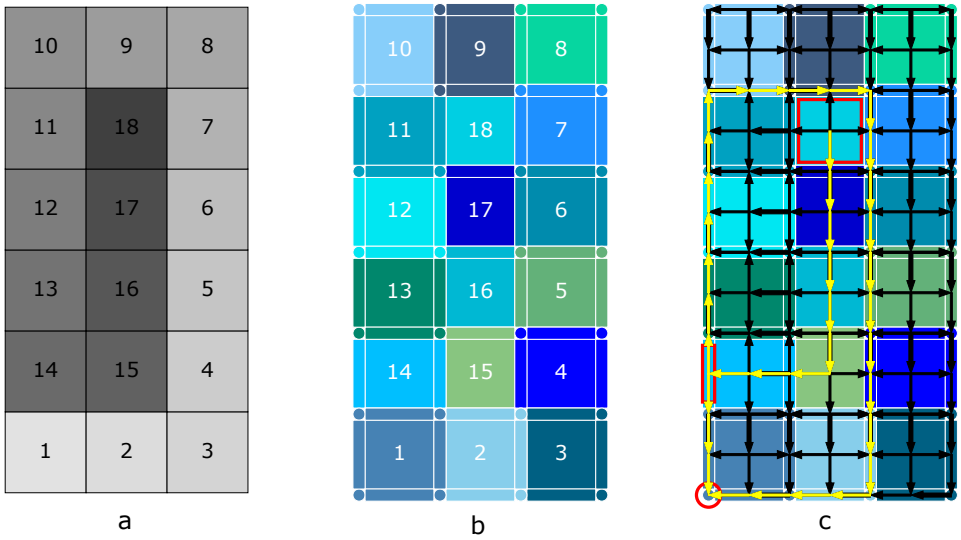


Figure 2 (a) Input volume; (b) Filtered cubical complex (c) Depiction of the $\mathcal{RWS}$ method. Black arrows show the matching graph. The yellow paths mark the alternating paths whose parity determine the boundary relations between the red critical cells. There is one nonzero relation: between the critical 2-cell with value 18 and the critical 1-cell with value 14, which itself has empty boundary. In this simple case the resulting boundary matrix is already reduced, so the persistence pairs are readily available. Typically the matrix has to be reduced first.

6 Streaming simplification scheme

In this section we describe our algorithmic scheme, which is an efficient streaming version of the $\mathcal{RWS}$ scheme. It yields a reduced representation encoding the same persistent homology as the input volume. More precisely, we output – to disk – information about the boundary matrix of the corresponding filtered discrete Morse complex. This boundary matrix is then reconstructed on disk and the persistent homology is computed using an existing matrix reduction algorithm. We focus on a high-level overview, noting that the actual implementation is intricate and contains 2500 lines of terse C++ code (not counting external libraries).

Algorithm 1 outlines the streaming simplification scheme. In overview, we independently simplify each stratum using the method outlined in the previous section and put global information back together. We now outline the algorithm and prove its correctness.

Correctness. We need a few new concepts. A **border-crossing path** is an alternating path which contains cells belonging to two or more internal strata. A **global matching** is the union of discrete Morse matchings computed separately on each stratum of a filtered cubical complex. A **global Discrete Morse complex** is the filtered discrete Morse complex arising from the global matching.

We propose a lemma and its three corollaries. Together they show that the information extracted from all slices is sufficient to reconstruct the boundary matrix of a global discrete Morse complex that captures the correct persistent homology of the entire original dataset.

▶ **Lemma 1** (Border Blocking Lemma). *Suppose an acyclic discrete Morse matching is computed for each block of each stratum of a filtered cubical complex. Consider a directed path* $p = (p_1, p_2, \ldots, p_i, \ldots, p_n)$ *in the matching graph* G *corresponding to the global matching. If any* p_i *belongs to a border stratum* B*, then the suffix* $(p_i, p_{i+1}, \ldots, p_n)$ *is contained in* B*.*

Algorithm 1 Streaming simplification.

Require: $\boldsymbol{V}$: volume on disk; $(w_1, w_2, \ldots, w_D)$: size of V; $\boldsymbol{h}$: maximum height of slice
Ensure: Boundary matrix on disk with the same persistent homology as V

for index i in $1 \ldots \lceil w_1/h \rceil$ **do**
  $n = h \prod_{d=2}^{D} w_i$
  D = read next n voxels of V from disk (less for last slice)
  C = filtered cubical complex of D representing the slice
  O = load 1-voxel tall overlap for each border of C
  update the values of border cells in C using extra information in O
  S = decompose C into an internal stratum and up to 2 border strata
  for stratum s of S **do**
    for block b in s **do**
      compute acyclic Morse matching within b
      mark critical cells in S
  for critical cell σ in C **do**
    if σ does not belong to the bottom border stratum of C **then**
      write value and dimension information: (ind(σ), val(σ), dim(σ)) to disk
    $p = \dim(\sigma)$
    T = critical $(p-1)$-cells reachable from σ by odd number of paths
    for critical $(p-1)$-cell τ in T **do**
      write boundary information: (ind(σ), ind(τ)) to disk
  unload all data from memory
sort the indices by corresponding value and dimension (on disk)
map these sorted indices to a contiguous range of indices (on disk)
save the filtered boundary matrix in appropriate sparse format (on disk)

Proof. We proceed by induction, with $p_i \in B$ by assumption. To show that $p_k \in B$ implies $p_{k+1} \in B$ for $i \leq k < n$, we consider two cases. Namely, there are two potential outgoing edges from p_k in G: to a proper face of p_k, which belongs to the border stratum B because each border stratum is a cubical complex; or to a *matched* (paired) proper coface of p_k, which belongs to the border stratum B because the corresponding matching is restricted to B by construction. In any case, p_{k+1} lies in B and by induction so does each p_j for $i \leq j \leq n$. ◀

We recall that the internal strata are generally not cubical complexes. This means that paths can lead from an internal cell to a border cell. Still, the following corollary reassures us that no cycles can be formed.

▶ **Corollary 1** (Global Acyclicity). *The global discrete Morse matching is acyclic.*

Proof. First, there are no cycles within each stratum by construction. To form a directed cycle spanning multiple strata, a path would have to go back and forth between internal and border strata. This is however impossible, since paths entering a border stratum remain inside this stratum, as shown in Lemma 1. ◀

With the above, it is easy to see that we preserve information about persistent homology.

▶ **Corollary 2** (Global Correctness). *The global discrete Morse complex encodes the same persistent homology.*

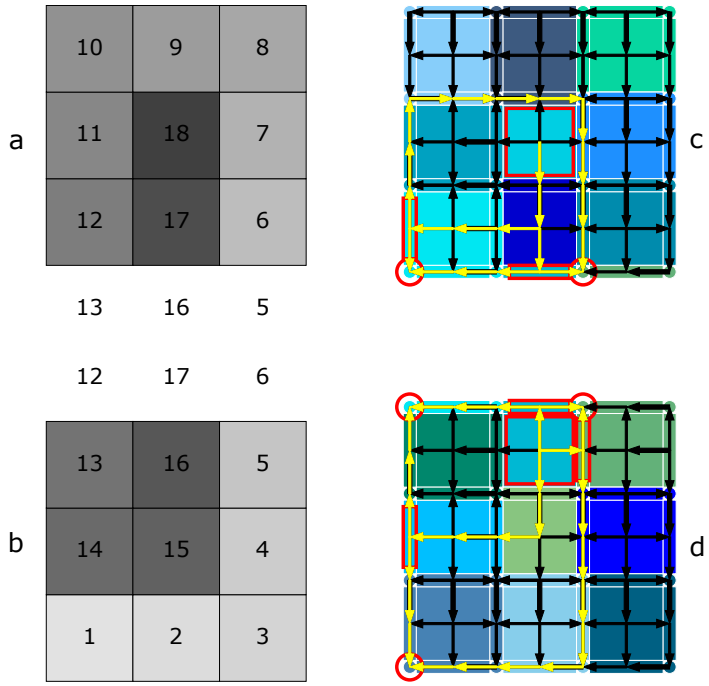


Figure 3 (a,b) Input split into two overlapping parts; (c,d) Simplification applied on each slice. The slices agree on the values and matching on the shared border. The path that crossed the border in Figure 2 is now split into *three* paths. One can construct the filtered boundary matrix with 8 columns and 13 nonzero entries to verify that, in particular, the 1-dimensional feature created in one slice and destroyed in another is correctly captured; also that the extra critical cells result only in features of zero persistence.

Proof. By Corollary 1, the global matching is a valid – if suboptimal – matching on the entire input filtered cubical complex. This reduces the proof to the case of the $\mathcal{RWS}$ approach we covered in the previous section. Therefore, the simplified complex encodes the same persistent homology (up to ephemeral pairs) as the original cubical filtration. ◀

Or scheme introduces extra critical cells, even if all the voxel values are unique. These extra cells allow us to stitch together discrete Morse complexes coming from different slices. More specifically, these extra cells split each border-crossing path into multiple non-crossing paths as shown in Figure 3. Instead of performing costly pruning at this stage [20, 10], we welcome these cells into the the final boundary matrix, reduce it, and simply discard the resulting ephemeral pairs from the resulting diagram.

Finally, we show that the information available in each slice is sufficient.

▶ **Corollary 3** (Slice Locality). *The boundary matrix of the global discrete Morse complex can be computed from information contained within each slice.*

Proof. Lemma 1 implies that there are no border-crossing paths. This means that all the boundary information contained in the resulting boundary matrix can be computed locally within in each slice. Additionally, the dimension and value of each critical cell is available within each slice. ◀

One subtlety: in a practical implementation the values and matching assigned to each border stratum must be consistent between the adjacent slices. To ensure this, we load the overlap, namely the extra layers of voxels, in line 5 of Algorithm 1.

Computational complexity. Computations are dominated by tracking the parity of alternating paths [33], which is done in line 16. Overall, a single slice is handled in $\Theta(v^3)$ worst case time [33], where v is the number of voxels in a slice. In our case, a dataset with v voxels divided into s slices, the worst-case complexity is $\Theta(s(\frac{v}{s})^3) = \Theta(\frac{v^3}{s^2})$. However, this worst-case behaviour is theoretical, and the experiments we report in Section 7 show roughly linear scaling for *all* practical datasets.

7 Technicalities

In this section we mention several technicalities which make our implementation efficient in practice. Balancing speed and memory usage was the key challenge in our streaming setup.

Encoding the information about cells. All auxiliary information about cells are stored in a cube-map format [36], which simply arranges cells as an array of size $2w_1 + 1, 2w_2 + 1, \ldots, 2w_D + 1$ for input of size $w_1, w_2, \ldots, w_D$. We use an efficient implementation of multidimensional arrays provided by the blitz++ library.

Global indexing of cells. Each slice needs to assign a globally unique index to each of its critical cells. Since additionally the indices of border cells must be consistent across slices, we simply use the global index of cells in the entire complex. To compute this, each slice must know its offset and the dimensions of the volume (except w_1). These indices are compactified in line 13 of Algorithm 1.

Computing and storing the matching. Focusing on $D \leq 3$ allows us to adapt the ProcessLowerStar procedure of $\mathcal{RWS}$. Since each block can have at most 27 cells, these computations are unlikely to be the performance bottleneck.

However, storing the matching could dominate memory usage, which was alleviated in [19] by using a succinct bit-level encoding of the matching graph. Our sliced setting allows for more relaxed memory management, so such techniques are not necessary. Instead we used a simpler encoding, exploiting the fact that each edge can only point at at most 6 directions.

Computing the parity of alternating paths. This is hidden in line 16 and is crucial for performance of the simplification part. In $D \geq 3$, the alternating paths can both split and merge, which complicates the algorithms compared to lower dimensions. One consequence is that the number of alternating paths between 3-cells and 2-cells can grow exponentially in input size – which actually occurs in practical datasets [19]. This makes simple enumeration prohibitive. However, the matching graph is an acyclic directed graph, allowing us to use a basic dynamic programming algorithm for counting the parity of alternating paths.

One important addition compared to existing implementations is the usage of the bit-tree data-structure developed for the PHAT library. We re-purpose it for compact storage and manipulation of indices of cells maintained during path-counting. This removes the main memory bottleneck present in [19], which used a red-black tree. It required significantly more storage in the worst case and is also generally slower.

Reconstructing boundary matrix and persistence pairs. During simplification we output partial boundary and filtration information on disk in lines 14 and 18. In lines 20–22, we reconstruct the full filtered boundary matrix using a simple external-memory (on disk) algorithm. We do this to avoid storing both the partial information and the full matrix at

the same time. In line 20 of Algorithm 1, we sort the indices of cells by their values, since we need a filtered boundary matrix. In line 20, we map global indices of cells into contiguous indices, as required by most matrix reduction packages. Already in line 13 we made sure that the information for each border cell is output from exactly one of the two slices. In line 22 we rearrange the data in the format required by matrix reduction software; usually each column is represented by the dimension of the cell, the size of the column followed by its nonzero indices. Additionally, after the matrix reduction, we transform the reduced matrix into the final persistence diagram in a similar way. All these simple computations are implemented using the STXXL [11] external algorithms library.

Lock-free parallelization scheme. Each slice is handled by one thread. More precisely, we maintain a thread-pool of a fixed size k, which allows k slices to be handled in parallel. Each thread loads its slice, processes it, and outputs partial results to one of k output buffers on disk and unloads the slice. This allows for simple, lock-free parallelism, circumventing the usual synchronization problems. We use the CTL thread-pool library.

Experiments We present experiments which compare the end-to-end performance of the proposed approach with existing solutions.

Datasets. We use datasets provided at the free Open Scientific Visualization Datasets (**OSVD**) repository. This repository contains several 3D voxel datasets, mostly coming from real-world applications. The sizes range from 41^3 to 4096^3, with one 900GB file of size $10240x7680x1536$ voxels. For comparison we used the 15 smallest datasets in the repository, with up to 150 millions voxels. We also test our method on images up to 8 billion voxels.

Benchmarked software. A recent paper [18] includes a comprehensive benchmark with a multitude of software packages. We restrict our benchmark to three fastest end-to-end alternative approaches: CubicalRipser, DIPHA and DiscreteMorseSandwich. **CubicalRipser** is a modern method [27] based on implicit boundary matrix representation. It is a single-threaded implementation. We used a newer version than in the original benchmark [18], and got significantly better results. **DIPHA** [3] implements [36] and explicitly stores and reduces the full boundary matrix. **DiscreteMorseSandwich** [18] is part of TTK [34] and implements the $\mathcal{RWS}$ scheme. It works both in cubical and triangulated setting.

Result consistency. Our approach uses the data interpretation described earlier in the paper, sometimes called the T interpretation [16]. We verified that the produced persistence diagrams are consistent with the results of CubicalRipser with appropriate options. DIPHA uses the dual V interpretation, which however encodes equivalent information [16]. As already noted in [27], DiscreteMorseSandwich returns different diagrams presumably due to an alternative data interpretation.

Hardware. We use a commodity laptop with an i7-1165G7@2.80GHz CPU with 5MB L2 cache, 8 logical cores, 32GB of RAM and Toshiba XG6 M.2 NVMe SSD. All software is compiled and run on Ubuntu 20.04.5 using g++ 10.3.0. All parallel implementation are run on 8 threads.

Performance comparison on smaller data. We now discuss results of the experiments. We start with a comparison with other software on smaller datasets. Then we turn to much larger data to understand performance characteristics and limits of our approach.

Figures 4 and 5 depict the speed and memory efficiency of the proposed method and existing implementations. Our goal was to provide an efficient method for massive voxel data running on commodity hardware. For small datasets the overhead related to streaming and external memory operations dominates the computations – which is a trade-off we made by focusing on massive data. In this case, the new version of CubicalRipser is the dominant solution. For *all* datasets larger than $256^3 \approx 10^6$ voxels, our approach is both faster and more memory efficient.

Speed. For large enough data, our approach achieves speed between 0.25 and 2 million voxels per second and is usually an order of magnitude faster than the fastest alternative. The computations scale linearly with size, and also depend on the complexity of the data, which we investigate in a moment.

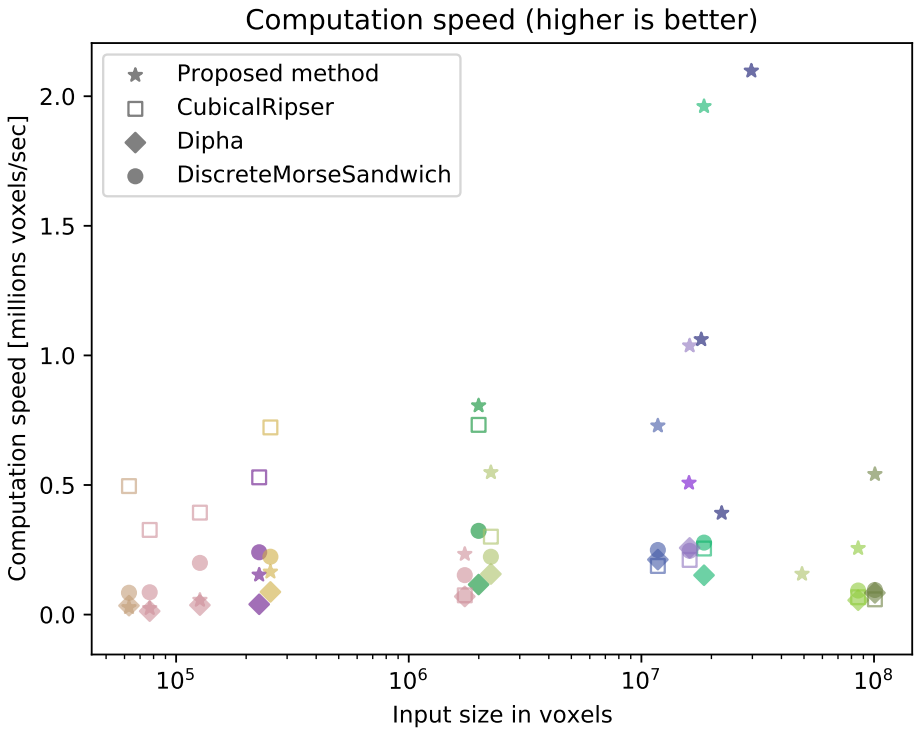


Figure 4 Speed comparison in millions of processed voxels per second. Methods are marked with different symbols. Random colors help distinguish between datasets. For datasets exceeding 1 million voxels, the proposed method offers the highest execution speed. In some cases competing software failed to finish in reasonable time, which explains why some points are missing.

Memory. For large enough data, the proposed approach achieves much smaller memory footprint. For the tested datasets the peak memory usage was kept below 3.5GB, whereas alternative approaches required > 24GB of memory. Since memory consumption was the obstacle preventing analysis of large volumes, the low memory usage is the main selling point of the new approach.

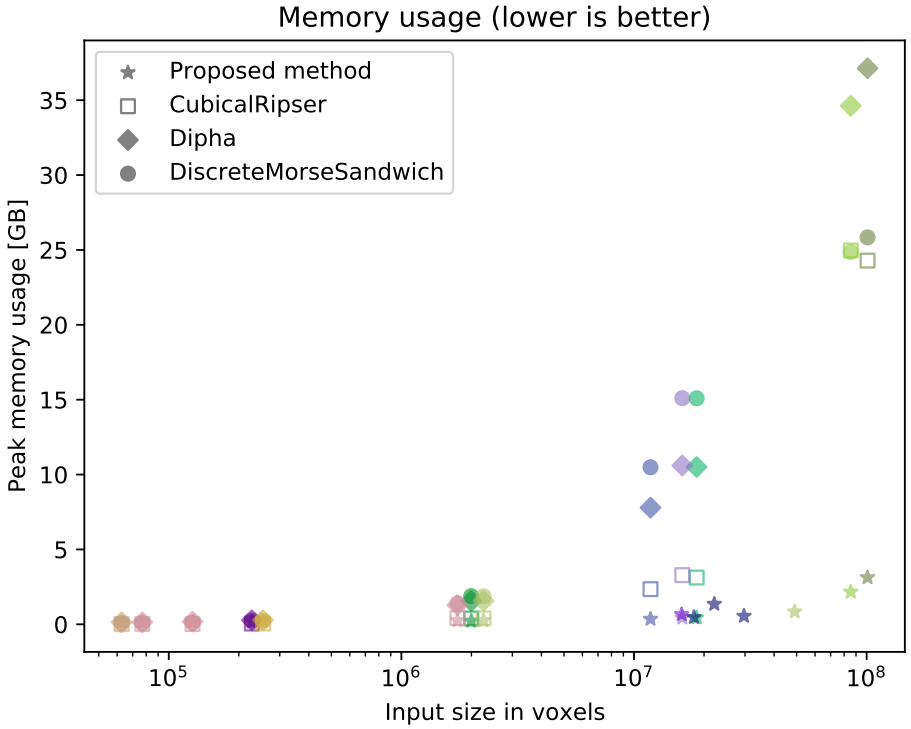


Figure 5 Peak memory usage comparison. For datasets exceeding 1 million voxels the proposed method offers significantly lower memory usage. Memory usage is negligible for smaller datasets.

Performance on large data. We turn to much larger datasets counted in billions of voxels – beyond the scope of the other approaches, at least on our test hardware. Results are summarized in Table 1.

The technicalities described in Section 7 aimed to balance memory usage and speed. In particular, storing boundary matrix information in memory would significantly increase the memory footprint. On the other hand, a slow on-disk implementation could impact the overall performance. On a more fundamental level, one big unknown was the number of extra critical cells our method will generate. They could easily overwhelm the computations or lead to huge boundary matrices. These experiments show that we found a reasonable balance and that memory usage was significantly decreased slowing things down.

We mention two data-points not collected in the benchmark. First, the final matrix reduction accounts only for a small portion of the execution time. Second, storage of the boundary matrix often dominates the memory usage. Therefore, further research into memory-efficient matrix reduction algorithms would benefit our implementation. The work reported in [5] is a step in this direction. We also mention that for some of the examples the memory usage could be further reduced by setting the slice size parameter to a smaller value.

Richtmyer–Meshkov instability. The last dataset in Table 1 is a snapshot of a 3D simulation of the Richtmyer–Meshkov instability [8]. It describes impulsive mixing of two different density fluids – and often leads to multi-scale behaviour exhibiting topological patterns. This particular file has size $2048 \times 2048 \times 1920$, roughly 8 billion voxels and 64 billion cells. Its full boundary matrix takes ≈ 1.5TB, and reducing it would require at least 3TB of RAM. Our method requires 150 times less allowing it to work on a regular laptop.

Table 1 Columns represent **file**: original input filename; **t[s]**: total time in seconds; **Mvox/s**: speed in millions of input voxels per second; **sim**: percentage of time spent in the simplification step; **B[M]**: millions of nonzero elements in the resulting boundary matrix; **c[M]**: millions of critical cells; **p[M]**: millions of persistence pairs; **m[GB]**: peak memory usage in GB.

file	t[s]	sim	Mvox/s	B[M]	c[M]	p[M]	m[GB]
vertebra_512x512x512	107	74%	1.2	20	7.6	2.8	1.3
zeiss_680x680x680	190	87%	1.7	17	7.7	0.4	2.2
prone_512x512x463	231	54%	0.5	73	29.5	12.9	3.1
neocortical_..._1464x1033x76	266	59%	0.4	81	32.1	11.9	3.1
present_492x492x442	290	53%	0.4	100	37.8	15.5	4.2
stent_512x512x174	291	92%	0.2	16	5.6	2.5	0.8
christmas_tree_512x499x512	333	42%	0.4	132	53.1	20.7	5.3
marmoset_..._1024x1024x314	662	40%	0.5	273	111.4	34.5	10.4
kingsnake_1024x1024x795	1345	62%	0.6	337	140.8	51.6	13.8
pawpawsaurus_958x646x1088	1809	30%	0.4	573	232.0	109.5	21.3
chameleon_1024x1024x1080	2152	48%	0.5	619	261.4	122.3	21.8
richtmyer_..._2048x2048x1920	11477	86%	0.7	828	292.5	88.4	22.3

8 Outlook

The main contribution of this work is a streaming preprocessing scheme which reduces the representation of voxel data without affecting its topology. In particular, it provably preserves persistent homology of the data. We combined our scheme with an existing boundary matrix reduction algorithm, yielding an end-to-end solution for persistent homology computations. Our experiments show that for large data our solution is the most efficient option.

> Our method achieves speed between 0.2 and $2 Mvox/s$ depending on input complexity. It handles complex data with 2048^3 voxels on a laptop.

We offer three interesting future directions the proposed scheme opens up:

- We can now handle multi-scale datasets of several billion voxels. Data coming from physical simulations, astrophysics and nanotechnology often exhibit multi-scale structure – and using persistent homology on such data is an exciting prospect.
- With little extra technical effort, we can stream huge data from a network location. This is important since raw voxel volumes of up to 900GB are readily available but copying and storing them can be a nuisance.
- Further progress in matrix reduction algorithms will benefit our approach. In particular, an external-memory matrix reduction algorithm would complement our scheme well.

References

1 Ulrich Bauer. Ripser: efficient computation of vietoris–rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, 2021.

2 Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization*, 2014.

3 Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology. In *2014 proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 31–38. SIAM, 2014.

4 Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat: Persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78:76–90, 2017. Algorithms and Software for Computational Topology. doi:10.1016/j.jsc.2016.03.008.

5 Ulrich Bauer, Talha Bin Masood, Barbara Giunti, Guillaume Houry, Michael Kerber, and Abhishek Rathod. Keeping it sparse: Computing persistent homology revised. *arXiv preprint arXiv:2211.09075*, 2022.

6 Paul Bendich, Herbert Edelsbrunner, and Michael Kerber. Computing robustness and persistence for images. *IEEE transactions on visualization and computer graphics*, 16(6):1251–1260, 2010.

7 Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European workshop on computational geometry*, volume 11, pages 197–200, 2011.

8 Ronald H Cohen, William P Dannevik, Andris M Dimits, Donald E Eliason, Arthur A Mirin, Ye Zhou, David H Porter, and Paul R Woodward. Three-dimensional simulation of a richtmyer–meshkov instability with a two-scale initial perturbation. *Physics of Fluids*, 14(10):3692–3709, 2002.

9 Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian Sheppard. Skeletonization and partitioning of digital images using discrete morse theory. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):654–666, 2014.

10 Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian Sheppard. Skeletonization and partitioning of digital images using discrete morse theory. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):654–666, 2015.

11 Roman Dementiev, Lutz Kettner, and Peter Sanders. Stxxl: standard template library for xxl data sets. *Software: Practice and Experience*, 38(6):589–637, 2008.

12 Pawel Dlotko. Cubical complex. In *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. URL: `http://gudhi.gforge.inria.fr/doc/latest/group__cubical__complex.html`.

13 Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

14 Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceedings 41st annual symposium on foundations of computer science*, pages 454–463. IEEE, 2000.

15 Robin Forman. A user's guide to discrete morse theory. *Séminaire Lotharingien de Combinatoire [electronic only]*, 48:B48c–35, 2002.

16 Adélie Garin, Teresa Heiss, Kelly Maggs, Bea Bleile, and Vanessa Robins. Duality in persistent homology of images. *arXiv preprint arXiv:2005.04597*, 2020.

17 Charles Gueunet, Pierre Fortin, Julien Jomier, and Julien Tierny. Contour forests: Fast multi-threaded augmented contour trees. In *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 85–92. IEEE, 2016.

18 Pierre Guillou, Jules Vidal, and Julien Tierny. Discrete morse sandwich: Fast computation of persistence diagrams for scalar data–an algorithm and a benchmark. *arXiv preprint arXiv:2206.13932*, 2022.

19 David Günther, Jan Reininghaus, Hubert Wagner, and Ingrid Hotz. Efficient computation of 3D morse–smale complexes and persistent homology using discrete morse theory. *The Visual Computer*, pages 1–11, 2012.

20 Attila Gyulassy, Peer-Timo Bremer, Bernd Hamann, and Valerio Pascucci. A practical approach to morse-smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6), 2008.

21 Teresa Heiss and Hubert Wagner. Streaming algorithm for Euler characteristic curves of multidimensional images. In Michael Felsberg, Anders Heyden, and Norbert Krüger, editors, *Computer Analysis of Images and Patterns - 17th International Conference, CAIP*, volume 10424 of *Lecture Notes in Computer Science*, pages 397–409. Springer, 2017. `doi:10.1007/978-3-319-64689-3_32`.

22 G. Henselman and R. Ghrist. Matroid Filtrations and Computational Persistent Homology. *ArXiv e-prints*, June 2016. `arXiv:1606.00199`.

23 Xiaoling Hu, Fuxin Li, Dimitris Samaras, and Chao Chen. Topology-preserving deep image segmentation. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

24 Xiaoling Hu, Yusu Wang, Li Fuxin, Dimitris Samaras, and Chao Chen. Topology-aware segmentation using discrete morse theory. *arXiv preprint arXiv:2103.09992*, 2021.

25 Michael Joswig and Marc E Pfetsch. Computing optimal morse matchings. *SIAM Journal on Discrete Mathematics*, 20(1):11–25, 2006.

26 Tomasz Kaczynski, Konstantin Mischaikow, and Marian Mrozek. *Computational Homology*. Springer-Verlag, New York, 2004.

27 Shizuo Kaji, Takeki Sudo, and Kazushi Ahara. Cubical ripser: Software for computing persistent homology of image and volume data. *arXiv preprint arXiv:2005.12692*, 2020.

28 Kevin Knudson and Bei Wang. Discrete stratified morse theory: Algorithms and a user's guide. *arXiv preprint arXiv:1801.03183*, 2018.

29 Jonathan Shewchuk Martin Isenburg. Streaming connected component computation for trillion voxel images. In *Workshop on Massive Data Algorithmics*, 2009.

30 Konstantin Mischaikow and Marian Mrozek. Chaos in the lorenz equations: a computer-assisted proof. *Bulletin of the American Mathematical Society*, 32(1):66–72, 1995.

31 Konstantin Mischaikow and Vidit Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry*, 50(2):330–353, 2013.

32 Arnur Nigmetov and Dmitriy Morozov. Local-global merge tree computation with local exchanges. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

33 Vanessa Robins, Peter John Wood, and Adrian P Sheppard. Theory and algorithms for constructing discrete morse complexes from grayscale digital images. *IEEE Transactions on pattern analysis and machine intelligence*, 33(8):1646–1658, 2011.

34 Julien Tierny, Guillaume Favelier, Joshua A Levine, Charles Gueunet, and Michael Michaux. The topology toolkit. *IEEE transactions on visualization and computer graphics*, 24(1):832–842, 2017.

35 Alessandro Verri, Claudio Uras, Patrizio Frosini, and Massimo Ferri. On the use of size functions for shape analysis. *Biological cybernetics*, 70(2):99–107, 1993.

36 Hubert Wagner, Chao Chen, and Erald Vuçini. Efficient computation of persistent homology for cubical data. In *Workshop on Topology-based Methods in Data Analysis and Visualization*, 2011.

37 Fan Wang, Saarthak Kapse, Steven Liu, Prateek Prasanna, and Chao Chen. Topotxr: A topological biomarker for predicting treatment response in breast cancer. In *International Conference on Information Processing in Medical Imaging*, pages 386–397. Springer, 2021.

38 Fan Wang, Hubert Wagner, and Chao Chen. Gpu computation of the euler characteristic curve for imaging data. In *38th International Symposium on Computational Geometry (SoCG 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.