VHS: a package for homological simplification of voxelized plant root data for skeletonization

Erin W. Chambers [*] Tao Ju[†] David Letscher[‡] Hannah Schreiber[§]

Dan Zeng[¶]

Abstract

In this work, we present VHS (**V**oxelized **H**omological **S**implification), a C++ package whose purpose is to de-noise voxelized data and output a topologically accurate simplified shape. In contrast to previous work on voxelized homological simplification tools, our main goal is offering a better starting point for computing curve skeletons for shape analysis. This goal necessitates additional simplification beyond what other packages provide, although our approach extends and improves prior work on heuristic methods which compute approximate solutions for the homological simplification problem. Our tool is designed for and tested on voxelized plant roots, although it is potentially useful beyond this data set. While the homological simplification problem is NP-hard in general, our package is able to simplify almost all of the topological noise when used on data from plant root systems. Compared with existing simplification tools, our method strikes a better balance between topological simplicity and geometric accuracy, resulting in higher usability of the resulting skeletons. Our code is publicly available at `https://github.com/davidletscher/VHS/`.

1 Introduction

Plant root systems are of particular interest to plant biologists, since the geometry and architecture of root systems play a key role in determining the genetic basis which can improve crop yield and identify environmental impacts [16]. However, the "hidden" nature of roots make them much harder to study than the above-ground portion of the plant. Fortunately, recent advances in 3D imaging have led to considerable advances in computational analysis of roots [23, 27, 28]. However, most image-based root phenotyping methods only compute overall traits such as the volume, depth, convex hull volume, total root length, and root number [13, 14, 18, 33]. Though useful, these traits which are aggregated over the whole root system do not capture the branching structure or the hierarchical organization of individual roots. Some recent work has given more detailed traits for maize crown roots [17], which gave evidence that geometric traits of the root are quite compelling to

[*]University of Notre Dame
[†]Washington University in St. Louis
[‡]St. Louis University
[§]INRIA
[¶]Washington University in St. Louis

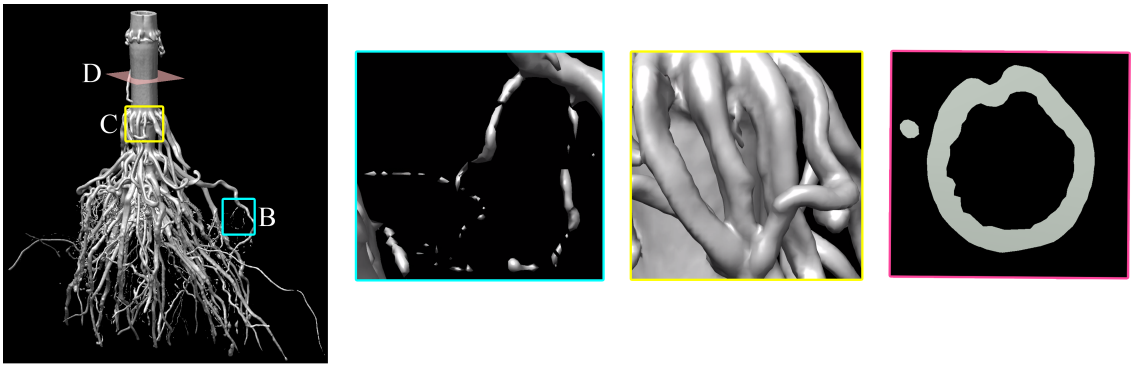

Figure 1: (A) A corn root before simplification, containing thousands of topological errors including (B) disconnected components, (C) handles, and (D) voids (shown in cross-section).

the plant science domain; however, a major inherent challenge in this system came from occlusion, as the roots were too densely packed roots to give any results beyond the very top of the root system.

To extract biologically relevant properties of the roots from the voxel set that can consider the entire root structure's geometry, we propose and focus on *curve skeletons*, or networks of 3D curves lying centered to the shape. Computing curve skeletons from 3D voxelized shapes has been extensively studied in the computer graphics community [30], and such skeletons have more recently played enabling roles in recent works on imaging-based root system analysis [33, 24, 28].

A key challenge in computing root skeletons is that the input voxel shape often contains numerous errors that prevent the skeleton from accurately capturing the topology and geometry of the roots. These errors may be due to the imaging process, such as instrumentation error, noise, insufficient contrast or resolution, or arise from the segmentation algorithm. We have observed two typical types of errors on a segmented root system that could significantly impact the utility of the resulting skeleton for downstream biological analysis. The first type is topological, such as spurious components, handles, and voids (see Figure 1 B,C, D). Such errors would lead to skeletons inheriting the same set of topological errors and hence failing to capture the true branching structure of the root system. The second type is geometric, and most notably concerns the thicker root areas near the stem. This portion of the root often has a hardened wall surrounding an interior space with lower intensity, and the resulting segmentation would appear to have a hollow space surrounded by a shell (see Figure 1 D). Note that a hollow space may not be a topological void, since it can be (and often is) connected to the exterior space via openings on the shell (see Figure 9). Without filling these hollow spaces in the segmentation, the resulting curve skeleton will not be able to capture the geometry of these roots and will instead consist of either spurious curves or two-dimensional surfaces (depending on the skeletonization algorithm) that lie completely in the shell space. As examples, the left column of Figure 2 shows the skeletons computed using [38, 37] for three corn root systems segmented from CT volumes using intensity thresholding. Observe that these skeletons contain numerous topological errors (e.g., broken curve segments, two-dimensional sheets surrounding voids), and the thicker roots (e.g., the main stem) are not captured by distinguishable skeleton curves.

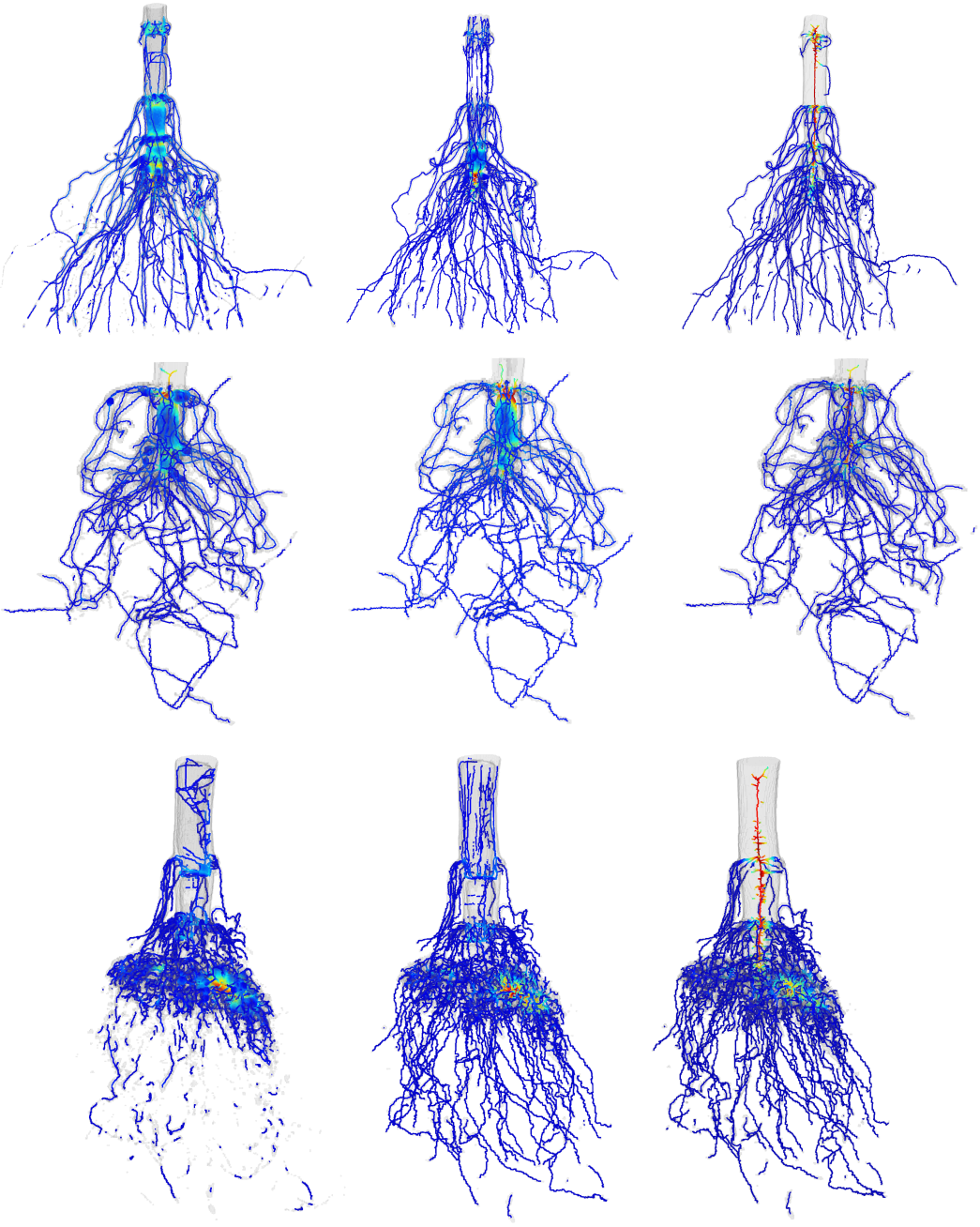

Figure 2: Left column: Corn roots before simplification superimposed with its resulting (messy) skeleton. Middle column: The result after simplification with prior methods, which perform only additive simplifications [7]. Right column: The result after applying VHS to the input root. These examples were produced using thresholding on a voxelized input root to segment, and then skeletonized via the package developed in [37]; color indicates the distance from the skeleton to the shape boundary (larger values in red).

To facilitate better analysis of root shapes, we provide and demonstrate a software package, VHS, to remove errors in the segmentation so that the resulting simplified shape produces cleaner and more usable skeletons for downstream analysis. Our package addresses both types of errors mentioned above - redundant topological features and hollow spaces within thick roots. Persistent homology is the main tool we use for simplifying topological features. It is most commonly used for measurement of topological features in a space, as it can detect underlying structure in the data even in the presence of noise; see for example [22] or other references for an in-depth introduction to its many uses. Here, we use persistent homology to identify features that are likely to be resulting from noise or sampling error, and then compute candidate cuts and fills (regions which we remove or add to the shape) with the goal of a simplified shape that is both topologically accurate and geometrically similar to the original shape. However, even when the shape is simplified, the resulting skeletons are not always "root-like". See for example Figure 2, where shape in the middle column actually has fairly good topological accuracy, but when compared to the right column does not skeletonize nicely to something that looks like a root.

As mentioned above, one issue is hollow spaces that are not topological voids (which we name *pseudo-voids*). To fill these, we devise a watershed-based algorithm that identifies and fills voids that appear in a filtration of the input. In comparison with existing methods that solely focus on topological simplification [40, 7], our method strikes a better balance between the topological simplicity and the ability of the skeleton in characterizing root geometry. A visual comparison between our method and a previous simplification method is shown in Figure 2. Note that the skeletons computed from the volume processed by our method (right column) retains roughly the same the topological simplicity of those computed from a previous method [7] (middle column) while doing a much better job in capturing the stem by a skeleton curve. A more quantitative analysis of these and other data is presented in Section 5.

In this work, we make extensive use of a pair of methods, VoxelCore [37] and Erosion Thickness [38], which were developed in the computer graphics community; when combined, they compute medial axis based topologically accurate skeletons for voxelized shapes. See Figure 2. We note that our simplification process is more broadly usable as a preprocessing step before skeletonizing on roots and demonstrate similar improvements to the resulting skeletons computed using Giaroots [33], a skeletonization package tailor made for plant root systems. See Figure 3, as well as Figure 9 in the Experiments section for a more detailed discussion of the improvement and technical details. While our method is designed for and tested on plant root segmentations, it is well suited for processing voxelized segmentations of other tree-like shapes as well, such as blood vessels, lung airways, and neuron networks.

1.1 Related Work

From the plant science domain, there is relatively little work on this type of shape analysis, as the ability to image such roots is new. To the best of our knowledge, DynamicRoots is the only published and validated root phenotyping method that produces a full branching hierarchy and root traits associated with each hierarchy level in 3-dimensions [29]. However, this tool is designed to work only on time-series root systems of young plants grown in transparent gel, and hence have relatively simple geometry and structure. As a result, this tool is able to obtain high quality 3D voxelized reconstructions. While in theory DynamicRoots can process any segmented 3D root image, its accuracy suffers from number of topological errors in the segmentation. Although such errors are scarce the relatively simple and young roots they focus on, they are abundant in 3D

Figure 3: The same corn root's skeleton, first using an older simplification process [7] (left) and then using our simplification process (right). Skeletons were computed the package Giaroots [33]. Note the overall simplification in the skeleton, especially near the central core of the root where segmentation is most difficult. See also Figure 9 for a more detailed discussion and zoomed in example the demonstrates the improvement.

images of the more complex root systems we study. In addition, DynamicRoots requires a time series to obtain the correct root hierarchy, which is not obtainable in the maize roots which are dug up and scanned, so the software simply will not work on our data. We refer to [41] for a more detailed comparison of their methods to the use of skeletons from the plant sciences domain perspective, as their tool will not work on our data sets.

From the more mathematical side, a large number of packages that have been developed for computing topological persistence, but much less work has been done in topological simplification using persistence, which is necessity in this data set. For intensity fields in two and three dimensions, there has been a variety of work on simplifying the sequence of iso-surfaces, for example [4] and [32]. For simplifying a single voxelized shape, TTK [31] can repair connectivity issues and fill voids but cannot repair loops or handles in the data. There is also work on simplifying shapes and skeletons using tools from discrete Morse theory [25, 8], which have close connections to the tools from persistence; however, the skeletons computed in [8] were focused on porous substances and work to maintain the same topology, rather than attempting to simplify to a tree.

Previously, a subset of the authors of this work were able to deal with general topological errors in cubical data, but the prior implementation would only grow the shape [7], adding what we term "fills". In this earlier work, we were unable to perform deletions (or "cuts") in the shape to remove topological noise. Other approaches to topological cuts and fills include more combinatorial approaches, where the problem is modified as a graph problem solvable by Steiner tree approximations [40]; in the conversion to a graph based approach, collections of cuts and fills are grouped together and assigned scores, and the minimum weight approximate solutions are computed. In contrast, our approach is more persistence based, allowing finer grained cuts and fills.

A key element in our approach is using persistence to identify generators, or candidate elements

to be added or removed in order to simplify the topology. Computing geometrically nice generators is a well-studied problem in the simplicial domain; for example, in many settings algorithms are able to identify loops in a mesh which can then be cut or filled, in an effort to improve the overall reconstruction of the shape [9, 36]. There is much less work done in this area for voxelized data; to the best of our knowledge, the package TTK [32, 31] is the only one to address voxelized or cubical data, and as noted above only focuses on connectivity and void filling (fixing H_0 and not higher homology errors).

Since submitting this paper, the authors of this work have published and shared a toolkit, TopoRoot [41], which includes both VHS and another method for topological simplification as optional ways to denoise and simplify plants. The other methods was developed in parallel to this one, and chooses cuts and fills based upon a transformation of the shape to a Node-Weighted Steiner Tree problem, followed by using solvers for that problem to heuristically compute a solution [39]. We discuss our method's results compared to the Steiner Tree approach in more detail at the end of Section 5, but in short, that method is faster but of lesser quality in terms of the calculated simplification and skeleton.

1.2 Our Contribution

Our approach to topological simplification follows previous work [7] and phrases the denoising in terms of the homological simplification problem [1]. Here, this means we have a shape S that we want to simplify as well as a "core" of the shape and a "neighborhood". The core consists of voxels that we are certain are part of the shape, generally defined as especially high intensity voxels which are near to other high intensity voxels; intuitively, it is clear that these should not be removed. The neighborhood consists of voxels that could be part of the shape and could be added; in our application, this is generally defined as nearby voxels as well as some lower intensity voxels. The goal is to modify the initial shape by adding and removing voxels (which we term cuts and fills) so that it only the resulting shape is a solution to the homological simplification problem or close to one. Our preliminary implementation [7] approached this problem by only adding voxels to the shape to eliminate certain types of topological noise, whereas in this work we remove topological noise by adding and removing voxels to repair topological noise while remaining geometrically close to the original shape. Our package achieves an approximate solution to this more difficult simplification problem. In addition, we address certain artifacts, which we term "pseudo-holes", that are created in the process of homological simplification that cause problems when skeletonizing.

We begin in Section 2 with an overview of definitions and background necessary for the paper. In Section 3, we give the theoretical overview of our simplification process, including a discussion of the types of errors encountered and how to simplify each. In Section 4, we outline our implementation and the algorithm engineering necessary for building our package. Finally, we discuss the results of our implementation in Section 5, and conclude by discussing potential applications and future direction in Section 6.

2 Background and definitions

2.1 Voxelized Shapes

In computer graphics, a common representation for 3-dimensional data is obtained by dividing $\mathbb{R}^3$ into a cubical lattice where each cube or *voxel* is given a real number or *intensity*. These intensities

can be represented by a function $f : \mathbb{Z}^3 \to \mathbb{R}$. A *voxelized shape* is obtained by gluing the voxels with intensity greater that a threshold together. This representation is generally how MRI and CT data are stored, and hence is important in many application domains.

Connectivity. When forming a shape from the voxels there are two common ways to glue them together, referred to as 6-connected and 26-connected. Under 6-connectivity, two voxels are glued together if they share a square face in common. Note that if voxels are "diagonal" to each other they will not always be connected together. Under 26-connectivity, two voxels are glued together if they overlap in a common square, edge or vertex. These two different ways can result in shapes with different topology, so it is important to be specific how you are attaching the voxels. In this paper, unless specifically stated otherwise, we assume 6-connectivity. However, we must still address 26-connectivity, since a shape is defined using 6-connectivity will have complement that is 26-connected; see below.

Dual Skeleton. The dual skeleton of a voxelized shape X is a cubical complex whose vertices correspond to each voxel in X. Two vertices are connected by an edge, if the two corresponding voxels are 6-connected. Any higher dimensional cell exists in the skeleton if its boundary exists in the skeleton. The dual skeleton has the useful property that is captures the homology of X, since it is a strong deformation retract of the shape.

Collar. The *collar* of a voxelized shape X is the set of voxels not contained in X but 26-connected with at least one voxel of X. We define a *void connected* component of the collar as the set of all its 26-connected components "surrounding" the same internal void of X or the exterior of X (In other words, if two voxels can be connected together without crossing a voxel of X, they are included.)

2.2 Homology and Persistence

We give a brief overview of homology and persistence here, and refer to recent books [22, 10] for a more detailed introduction to these topics. All of our homology calculations will be done using *cubical complexes*, see [35] for discussions specific to this context. Essentially, a cubical complex consists of a collection of cells: vertices, edges, squares and cubes on a cubical lattice.

Homology. Homology is defined in terms of collections of cells in a cubical complex. For the purposes of this paper, we are working with $\mathbb{Z}_2$ homology, defined as follows. A d-chain is a collection of d-dimensional cells; the set of all d-chains for a cubical complex $\mathbb{K}$ form a vector space over $\mathbb{Z}_2$ and are denoted $C_d(\mathbb{K})$. For $\alpha \in C_d(\mathbb{K})$, we can define its boundary $\partial\alpha$ as the set of $(d-1)$-cells that appear in the boundary of an odd number of cells in α. There are two important subspaces of $C_d(\mathbb{K})$: the *cycles*, $Z_k(\mathbb{K})$, which are the chains with empty boundary; and the *boundaries*, $B_k(\mathbb{K})$, which are the boundaries of $(k+1)$-chains. The *homology group* is defined as

$$H_k(\mathbb{K}) = {}^{Z_k(\mathbb{K})}\!/_{B_k(\mathbb{K})},$$

where the quotient is defined algebraically. The k^{th} *Betti number*, $\beta_k(\mathbb{K})$, is defined to be the rank of $H_k\mathbb{K}$; intuitively, since homology groups mod out by boundaries, this rank represents the number of non-trivial topological features: connected components for dimension 0, cycles in dimension 1,

voids in dimension 2, etc. As the boundary map can be written in matrix form, all homology calculations can then be done using linear algebra tools.

Persistence. To define *persistent homology*, we start with a sequence of spaces, or *filtration*,

$$\mathbb{K}_0 \subset \mathbb{K}_1 \subset \cdots \subset \mathbb{K}_n.$$

Note that for $i \leq j$ there is a natural inclusion $C_k(\mathbb{K}_i) \subseteq C_k(\mathbb{K}_j)$. This enables us to define the persistent homology group $H_k^p(\mathbb{K}_i)$ as $Z_k(\mathbb{K}_i)/B_k(\mathbb{K}_{i+p})$; intuitively, this is the set of k-dimensional non-trivial topological features (cycles for H_1, voids for H_2, etc.) which exist in $\mathbb{K}_i$ and are still present in $\mathbb{K}_{i+p}$. The *persistent Betti number*, $\beta_k^p(\mathbb{K}_i)$ is the dimension of the vector space $H_k^p(\mathbb{K}_i)$. This Betti number measures the number of k-dimensional features of $\mathbb{K}_i$ that are still present in $\mathbb{K}_{i+p}$. In principle, this makes persistence a reasonable way to quantitatively measure how much the topology changes over the course of the filtration, as well as identify which features are longer lived and hence less likely present due only to noise or error.

Reduction Algorithm. To compute a summary of how the topology changes over the course of a filtration, the standard algorithm [11] computes a *barcode* by reducing the boundary matrix M to obtain a basis for the boundary group. Initially, to compute k-dimensional persistent homology, the columns correspond to $k+1$-cells, where the i^{th} column is the i^{th} cell to appear in the filtration, and the rows correspond to all k-cells, again in the order of filtration. Cell (i, j) then initially has a non-zero value if and only if the j^{th} k-cell is in the boundary of the i^{th} $(k+1)$-cell. We will be working with $\mathbb{Z}_2$ coefficients, so every non-zero value is a 1. Let the *pivot* of a column c with at least one 1 value be the last index of c having value 1. Then the reduction algorithm performs only left-to-right column additions such that in the reduced form of M all pivots have different indices. Now, if the i^{th} column has a pivot j, then the content of this column corresponds to a representative of the cycle class born at time j and dying at time i. The worst case complexity of this computation is cubical, but it has been observed to perform much better in practice; see for example [26] for a discussion of this in practice.

2.3 Homological Simplification

Consider a pair of spaces, $C \subset N$. The homological simplification problem asks the following: can the persistent homology $H(C \hookrightarrow N)$ be realized as the homology of some complex X, where $C \subseteq X \subseteq N$? More formally, we want the p^{th} Betti number $\beta_p(X)$ to be equal to $\beta_p(C \hookrightarrow N)$ for every p.

For filtrations of closed and orientable 2-manifolds, the homological simplification problem is solvable [12]. In fact, [12] solves a more general problem of finding an ϵ-simplification in a filtration, but fails to apply to 3-manifolds or higher since such spaces do not always have ϵ-simplifications. However, this is not the case for more general settings; Attali et. al [1] show that the homological simplification problem is NP-complete even for simplicial complexes embedded in $\mathbb{R}^3$.

3 Simplifying Voxelized Shapes

We begin by outlining our theoretical process of simplification in this section. This includes both homological simplification in Section 3.1, where we discuss the kinds of topological inaccuracies in

this data as well as how to repair them, as well as the problem of pseudo-voids in roots structures in Section 3.3, which are more specific to our data set and skeletonization, and which require additional handling even after topological simplification.

3.1 Homological simplification

Consider a voxel intensity function f extracted from our input data. Our initial choice of shape is made by choosing all voxels with intensity larger than a chosen threshold. Topological errors can arise in multiple ways, and such thresholding is notoriously error prone. In particular, even a small increase or decrease in the threshold can cause significant topological features to appear or disappear. Similarly, if the shape is geometrically grown or shrunk slightly, even though it remains geometrically close we can see significant topological changes.

This setting naturally gives us a 2-parameter family of voxelized shapes $X_{t,r}$ defined as follows. We start with with all voxels with intensity larger than t. For positive r, we then expand the shape to include all voxels within distance r of at least one voxel meeting the intensity threshold. For negative r, we remove voxels within a distance r of the boundary. While this 2-parameter family naturally captures the range of possible errors and the theory of persistence can be extended to such filtrations, persistent homology calculations for such 2-dimensional filtrations are inherently more complex [6] than for one dimension.

We will avoid these complexities since we are not trying to simplify all of the shapes in the filtration. Recall that our goal is to find a topological simplification of $S = X_{t,0}$. Given an intensity range $t \pm \epsilon$ and a small radius δ for the scale of geometric errors, we set up the following simpler filtration. We define the *core*, C, as the set of voxels that we determine must be part of our simplified shape. For example, $C = X_{t+\epsilon,-\delta}$, the set of all voxels that have a neighborhood with slightly higher intensities. Similarly, we define the *neighborhood*, N, as voxels that we could add to the shape, which we identify as likely candidates for some reason. For example, in this setting we can set $N = X_{t-\epsilon,\delta}$, which adds voxels that just missed being over the intensity threshold by a little as well as voxels that are nearby to the shape. Anything outside of the neighborhood should not be added to the shape. This gives us a pair of bounding shapes for our initial shape $C \subseteq S \subseteq N$.

For any choice of core and neighborhood, we then have a well-defined instance of the homological simplification problem as defined in the previous section. Our goal is now to modify the initial shape S such that we obtain a voxelized shape X with the properties:

- $C \subseteq X \subseteq N$
- $H_k(C) \twoheadrightarrow H_k(X) \hookrightarrow H_k(N)$ for all k

The second condition guarantees that $H_k(X)$ is the image of $H_k(C)$ in $H_k(N)$, which is the desired homology group $H_k(C \hookrightarrow N)$. This ensures that X only has the homological features that are shared by C and N. While our original shape S does lie in between, it is very unlikely to solve the homological simplification problem, since topological features appear as we grow or shrink.

Unfortunately, in general no such perfect X will always exist [20]. Furthermore, in a union of voxels there are additional technical complications when constructing a simplification. In particular, consider the following example: a voxel is added to the complex to simplify one topological feature, like filling a loop or merging two components, but in the process creates another loop. In two-dimensions this is easier to visualize, see Figure 4. In the example, the shape and the core have three components and no loops. If the grey pixel from the neighborhood is added then the resulting shape

is connected and has a single loop. The desired simplified shape should have a single component but no loops, which is not possible in this example given the problem constraints.

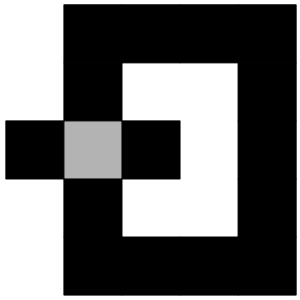

Figure 4: An example that cannot be simplified. C and S coincide and are the black pixels, while N also has the additional pixel in gray.

3.2 Types of errors and repairs

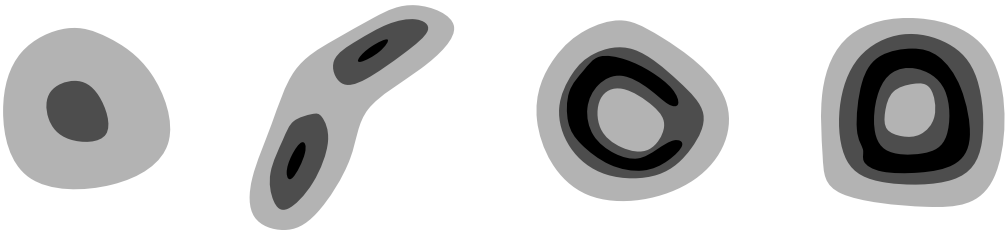

Figure 5: The four types of two-dimensional error, for several shapes with core, shape and neighborhood indicated in dark, medium, and light grey (respectively). From left to right: negative H_0, positive H_0, negative H_1 and positive H_1.

Given this issue, our algorithm instead will simply attempt to get as close as possible to X, even if an optimal solution does not exist. Our package applies two simple operations to S: the addition of voxels, which we call fills, and the removal of voxels, which we call cuts. The voxels in C are required to be part of any simplification of S and can therefore not be removed. Similarly, the voxels we can potentially add into S have to be in the neighborhood N.

To help visualize these types of errors in a simpler setting, see Figure 5. In two dimensions, 0-dimensional errors can be repaired by either removing a component of S, in the negative case or two components can be merged in the positive case. Similarly, 1-dimensional errors can be repaired by cutting a loop or filling in a hole. Note that some errors can be repaired by either cutting a loop or filling in a hole, see Figure 6.

As we are dealing with data embedded in $\mathbb{R}^3$, there are more cases to consider, but still only a few types of topological errors and repairs are possible:

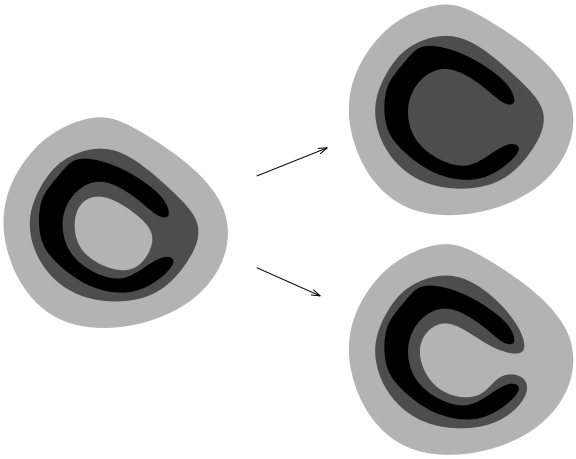

Figure 6: Some topological errors have multiple ways of being repaired. In this example, the shape on the left has a loop which must be either cut or filled; these options are shown the right, where the shape is either cut to remove the loop (bottom) or where a hole is filled (top).

- H_0 errors:
  - A component in C that is distinct in S but merges in N. This can be detected from distinct $\alpha, \beta \in H_0(C)$ that have different images in $H_0(S)$ but the same image in $H_0(N)$. This can be repaired performing a fill that is a path of voxels that connects the two components.
  - A component of S that does not contain any voxels of C. This appears as an element of $H_0(S)$ which is not in the image of $H_0(C)$. Each of these components is a cut and the shape can be repaired by removing all of them.
- H_1 errors:
  - A handle or loop in C that does not get filled in S but does in N. This can be detected as an element of $H_1(C)$ that maps non-trivially into $H_1(S)$ but becomes trivial in $H_1(N)$. A possible fill can be found by finding a 2-chain in the exterior of the shape whose boundary is a 1-cycle in $H_1(S) - H_1(C)$.
  - A handle or loop in S that is not in C. This appears as an element of $H_1(S)$ that is not in the image of $H_1(C)$. A cut through a new handle form in S can, potentially, remove this error.
- H_2 errors:
  - A void in S that get filled-in in N. This appears as an element of $H_2(S)$ that maps trivially into $H_2(N)$. Note that each void is a fill.
  - A void in S that is not completely enclosed in C. That happens when there is an element of $H_2(S)$ that is not in the image of $H_2(C)$. The formation of this void can be prevented by cutting a 2-chain from the shape.

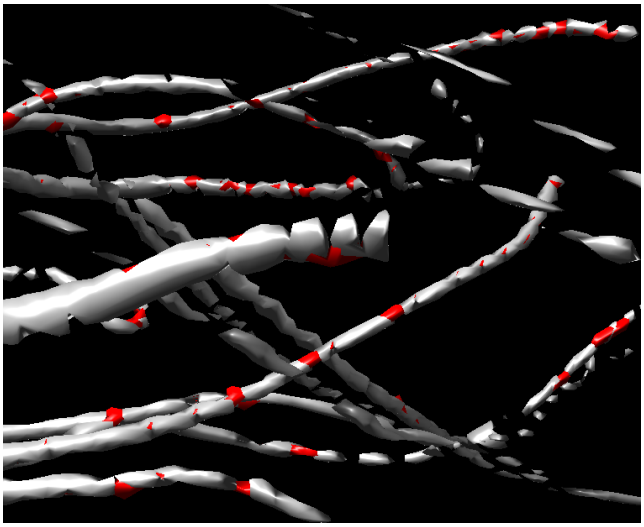

Figure 7: A portion of corn root, with potential fills shown in red.

In all of these cases, $H_k(S)$ either fails to inject into $H_k(N)$ or is not in the image of $H_k(C)$. If $H_k(S)$ does not inject then we want to add a set of voxels in N to S to kill an element in the kernel, and if $H_k(C)$ doesn't map surjectively than we want to remove voxels from S to reduce the rank of the kernel. Note that there are situations where elements of $H_k(S)$ are neither in the image of $H_k(C)$ and nor get injected in $H_k(N)$, so, there is the possibility that features could be corrected by either adding or removing voxels. See Figure 7 for a visualization of some errors and corresponding fills in corn root data, as computed in [7] and with our package.

3.3 Pseudo-voids

Even if we obtain a perfect solution to the homological simplification problem, certain types of data can still have artifacts present which make eventual skeletons poor representations of the underlying shape, with unnecessarily complex structure. The major issue is partial cavities in the stem of the root which have one or more punctures; we call those artifacts *pseudo-voids*. See Figure 8. Topologically speaking, this artifact is simply a birth in the H_2 homology under the distance filtration, yielding a void that was not present in the original shape but which appears when thickening. This is primarily an artifact of the data, as well as potential errors in it. Our primary use case, in the analysis of plant roots, shows this type of artifact quite frequently in or near the stem; other data sets of porous materials (such as bone) exhibit similar artifacts. In particular, for skeletonization, these voids cause many useless branches and loops, and resulting skeletons will not be centrally located in the stem for our data set. See also Figure 9 for an illustration of the skeletons.

Therefore, our package provides an option to pre-process and fill in pseudo-voids before simplification. To define a pseudo-void, we will work with an alternate filtration $\{Y_r\} = \cup_{x \in S} B_r(x)$ based on distance to the shape instead of voxel intensity. A *pseudo-void* appears each time there is a birth in $H_2(Y_r)$ for $r > 0$ and can be represented by a component of ∂Y_r. We are building

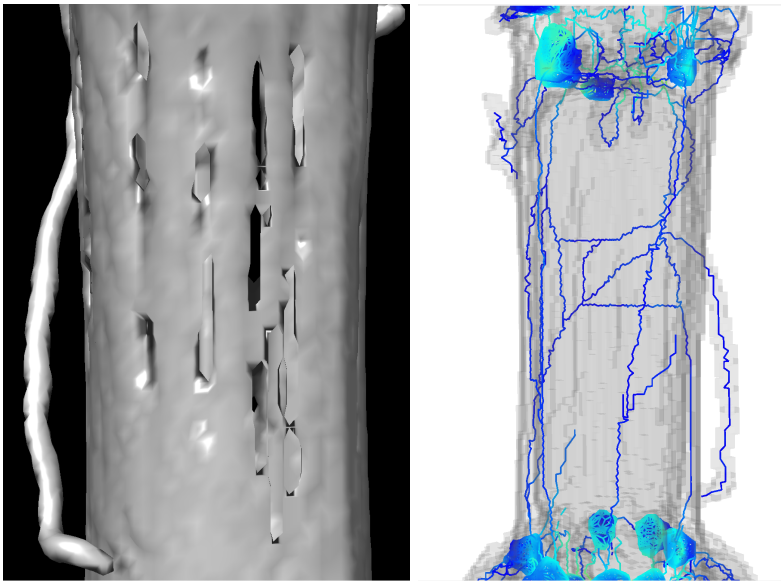

Figure 8: A root stem containing pseudo-voids, and the resulting (unsimplified) skeleton near those errors.

those components using a watershed algorithm that inflates the outside of the neighborhood, that is, building the complement of Y_r for a decreasing r. During the process, the border of the inflation will either hit the shape or another border of the inflation. Those "hit points" will represent the border of the pseudo-voids, which our algorithm can then precisely calculate and fill; see Section 4.1 for implementation details.

4 Implementation

In this section, we address the actual implementation of our theoretical approach. The source code is public and can be found at [15]. Consider an initial voxelized shape S with its core C and neighborhood N as defined in previous section. We begin by simplifying pseudo-voids via our novel approach, before topological simplification. After this preprocessing, we compute candidate cuts and fills in order to approximately solve the homological simplification problem. At a high level, our algorithm is a greedy two phases approach: generation of cuts and fills, and validation (or rejection) of those candidates. These two steps are then repeated until no further simplification is possible. The complexity of the approach lies in the details of how to ensure that new features are not unintentionally added to the shape, as well as the engineering process of how to make candidate generation and validation efficient.

See Algorithm 1 for a high level overview of our approach. The functionalities of all called methods are explained in the comments of the pseudo-algorithm and will be detailed in the following section. Note that the algorithm assumes that the stack of images slices the roots from top to bottom, that is, the first image (indexed by 0000) represents the top most cross-section of the main stem(s).

Algorithm 1 Simplification Algorithm

Input: Stack of grayscale PNG images representing a 3D root system

Output: Simplified images as a new stack of grayscale PNG images

1: $Voxels = (C, S, N) \leftarrow$ loadVoxelsFromStack ▷ Adds a voxel into C (core), S (intermediate shape) and N (neighborhood), depending on the given thresholds and its intensity

2: Add layer to $Voxels$ to close the hollow stem

3: **if** $fill\_pseudo\_voids$ is true **then**

4: realizePseudoVoids($Voxels$) ▷ Transforms pseudo-voids in S in real voids by adding voxels in N and S, see Section 4.1

5: fillAllVoids($Voxels$) ▷ Fills all voids in S independently of N by adding voxels in N and S, see Section 4.1

6: **else**

7: fillVoids($Voxels$) ▷ Fills voids in S completely contained in N by adding voxels in S and C, see Section 4.2

8: **end if**

9: Remove additional layer from line 2

10: removeComponents($Voxels$) ▷ Removes all connected components not containing core voxels by removing them from N and S, see Section 4.2

11: addOneVoxelCandidates($Voxels$) ▷ Generates H0 candidates containing only one voxel and if validated, add them to S and C, see Section 4.4

12: **while** $Voxels$ was modified **do**

13: addPositiveCandidates($Voxels$) ▷ Generates positive candidates and if validated, add them to S and C, see Section 4.2 and Section 4.3

14: removeNegativeCandidates($Voxels$) ▷ Generates negative candidates and if validated, removes them from N and S, see Section 4.2 and Section 4.3

15: **end while**

16: saveVoxelsToStack($Voxels$)

4.1 Simplifying pseudo-voids

We begin by considering line 4 and 5 of Algorithm 1, which are handling the pseudo-voids we defined in Section 3.3. Let N^+ be the neighborhood with where all voxels contained in a void are filled in. In this structure, we still contain pseudo-voids which will not be detected by homological simplification. Recall that such errors will seriously impact skeleton quality, as extra handles will be present in near the stem, and any resulting skeleton will not accurately represent root structure.

In order to generate possible simplifications that eliminate pseudo-voids, we start by computing the distance function from the shape for all voxels in N^+ using a Manhattan metric, which can be done in linear time. Then, we compute the starting voxels for the inflation in $O(n \log n)$ time: they will consist of the voxels outside but adjacent to N^+ and the local maxima of the distance function within N^+. Now, we proceed with the inflation from the starting voxels to the inside and whenever two fronts are colliding, we save the position instead of merging them. This can again be done in $O(n \log n)$ time. After computing all colliding components, we are filtering out those who do not touch the shape. The remaining components are then added into the shape to create an actual void in it. Finally, we fill all the voids in the shape in linear time. But different from the same function in the simplification process, we are filling in also voids in the shape that are not filled with voxels from the neighborhood. Therefore, the number of voids will systematically be zero after the filling, even if the optimal solution in the simplification problem is higher.

Note, that the actual volume of a voxel makes it difficult to obtain perfectly satisfying colliding components and also "falsify" some local maxima. Therefore, small handles or false pseudo-voids in form of a bump can be added to the shape. Luckily, they are usually small enough to be not too problematic in the overall skeleton, and some will even be simplified in our next homological simplification phase.

4.2 Computing candidate cuts and fills

We segregate candidate simplifications into two categories, cuts and fills, as described in Section 3.2. We then have two kinds of candidate generation methods: those which only need to be executed once and require no validation, and those producing possible candidates which are repeated until they produce no candidates anymore after validating and applying the proposed changes.

The first category of generation methods are the simplest to handle:

- For H_2-candidate fills: we can detect every void in the shape and fill them (if the neighborhood makes it possible) in near linear time. This corresponds to line 7 in Algorithm 1 if pseudo-voids are ignored and line 5 otherwise.
- For H_0-candidates cuts: again, we can remove every connected component (not containing any core voxels) in near linear time. This corresponds to line 10 in Algorithm 1.

For these two cases, we know that no void or new connected component is created by any of the validated cuts or fills, so we need to call those two methods only once at the start of the simplification.

For the remaining more difficult types of potential simplifications, we perform a persistence based calculation to construct candidate simplifications. For the additive cases, we examine the kernel of $H_k(S) \to H_k(N)$. Any element of the kernel is a cycle $\alpha \in Z_k(S)$ that is not a boundary in $Z_k(S)$ but is a boundary in $Z_k(N)$. Our goal is to find a chain $x \in C_{k+1}(N)$ with $\partial x = \alpha$. Then the minimal set of voxels containing the chain x will be used as our candidate fill.

Similarly, a cut can be detected when the map $H_k(C) \to H_k(S)$ is not surjective. However, this is not as easy to detect when directly working with the complex. Instead, we use the complement of the shape, so that additive errors in the complement will give us negative candidates with respect to the original. Formally, to work with the complements we must apply both Poincare and Alexander duality [21]. This results in us examining the kernel of the map $H_{k+1}(N - S) \to H_{k+1}(N - C)$. The fills found in this kernel are precisely the cuts when working with the original spaces.

To implement these tests to both find the topological errors and candidate repairs, we must convert our voxel representations into the dual skeleton which is a cubical complex. This results in a filtration $\mathbb{K}_C \to \mathbb{K}_S \to \mathbb{K}_N$ where these are the cubical complexes corresponding to the dual skeletons of C, S and N, respectively. We can now apply persistent homology where this is a filtration with times 0, 1 and 2 for the core, shape and neighborhood. The features that we are interested in are those dying when entering N at time 2. The standard reduction algorithm then gives us representatives for each cycle class dying at time 2. However, in our case, we are less interested in what is dying than in what is killing them, since these are our candidates. In other words, we need to retrieve the cell chains whose boundaries correspond to the representatives given by the reduction algorithm. To do so, we replicate all column additions made on the boundary matrix M on the identity matrix I. As in M, the i^{th} column of I represents the i^{th} cell in the filtration, but this time the content of the column is the cell itself. We then have that the boundary of $I[i]$ is equal to $M[i]$ for each i, and this property is preserved while replicating the column additions. Note that in order to do so, we cannot use the usual optimisation methods for the boundary matrix reduction which otherwise allows to skip several column additions. After reduction, we obtain the desired chains of cells whose boundaries are in the reduced form of M. Finally, we map back those cell chains into the corresponding voxels; these will constitute the fills.

To construct the cuts, we do exactly the same operation described above but in the complement of the shape, using the observation about duality described above to justify this choice of representatives.

4.3 Cut and fill validation

We now have a viable set of cuts and fills to simplify the shape. However, there are several reasons that cuts and fills need to be validated to ensure they simplify topology without unintended side effects, as illustrated in 2D in 4. In 3D, we have additional issues, such as additional genus that can be introduced when filling in a hole. These unintended side effects are in fact one of the major obstructions to solving the homological simplification problem, as simplifications can cause further changes in a cascading effect.

In our experience, a more common issue occurs when thickening a chain to the set of voxels containing it. The union of voxels often contain extra topology where the fill fails to be a regular neighborhood of the chain. In these circumstances the union of the shape and the fill fails to be a manifold. It is worth noting that expanding these fills to ensure that intermediate shapes are always manifolds is often counter-productive as they cause the addition of too many voxels. Furthermore, when there are multiple, possibly interacting, cuts or fills, there is the possibility that several simplifications, once put together, can introduce several new higher dimensional features. A cut or fill might also kill more features than expected. If this occurs, we apply the simplification and, possibly, reduce the total number of simplifications necessary.

To test our cuts and fills, we use an overlay on top of the shape such that we can temporarily add or remove voxels from the shape and verify how it impacts the number of topological features. The

number of features are tested differently depending on their dimension. For 0-dimensional fills, it suffices to count locally how many different connected components in the shape the fill is touching. If a candidate fill is touching two different components in the shape, it means that they are getting merged and the number goes down. For 0-dimensional cuts, components cannot merge, but they can split, which is what we want to avoid. So, the connected component containing our cut needs to be reconstructed to verify that we are not dividing any components into multiple pieces.

The 1^{st} Betti number, i.e., the number of handles, can be computed more quickly using a well-known property of the Euler characteristic χ:

$$\chi = \beta_0 - \beta_1 + \beta_2.$$

Practically speaking, this means our algorithm also needs to keep track of χ, which we do by using a second well-known property of the Euler characteristic:

$$\chi = |V| - |E| + |F| - |B|,$$

where $|V|$ is the number of vertices, $|E|$ the number of edges, $|F|$ the number of squares and $|B|$ the number of cubes in the dual skeleton of the shape. Changes in these values when adding to or removing voxels from the shape can easily be verified by looking at their neighboring voxels.

Finally, the number of voids can be counted by observing how the number of void connected components of the collar is changing. For this purpose, we are constructing the 26-connected components of the collar using a classic union-find structure. Then we add up-going vertical "stripes" of voxels to the right-most voxel of each connected component to connect every component which are void connected.

Note that for fills, the number of connected components can only decrease and similarly, the number of voids can only go down for cuts. Despite this fact, we still need to compute the new number of components and voids at each stage, because the computation of β_1 depends upon them. The number of connected components after removal and the number of voids are quite heavy to compute, because it requires completely reconstructing the union-find structure for the collar or the shape again each time. So, we avoid doing so for each candidate individually, when possible, by testing them in batches. But the validation process remains still slower than the generation process.

Unfortunately, a single pass of candidate generation, validation and shape modification is not sufficient. Once one change is made to the shape, other candidates might become invalid, and it is possible that no further simplifications can be used. As a result, our code repeats the process of generating new candidates and adjusting the shape until no more changes to the shape are made. See lines 12-14 in Algorithm 1. In practice, we found that most of the topological noise is removed in early passes, and a significant portion of the time is spent in these later passes that make minor improvements to the shape.

4.4 Practical Considerations

In order to optimize the performance, we need to balance the time spent generating and validating candidates. For the size of shapes we are dealing with, running the candidate generation and validation on the entire shape can take too long, so our implementation breaks the shape into smaller sliding windows and performs candidate generation on each window. As most of the topological errors are small, we miss very few simplifications in this process. The smaller the windows, the

quicker candidate generation occurs in each, but the more windows that need to be processed. Through experimentation we have determined windows sizes that work well for our data sets.

Another optimization step that we perform is handling small simplifications first; see line 11 in Algorithm 1. A significant percentage of simplifications involve adding or removing a single voxel. So, in the first pass we use separate candidate generation that only looks for one voxel candidates. This fixes many errors, since connectivity issues can be repaired when two components are very close together. In addition, many voids in our test data sets consist of a single voxel, which requires only a small window to see.

Internally, the shape is stored in in three nested hash tables, one for the core voxels, another one for the intermediate shape and a final one for all neighborhood voxels. Additionally, we are storing the connected components of S and its complement in both a Union-Find-Delete structure [5] and a hash table. This enables a fast `contains` and `remove` method for the three shapes and the collar, as well as fast access to the connected components of S. Because we never need to "split" a component, the delete operation in the Union-Find-Delete structure is as fast as the `find` method of an hash table. For the persistence based calculations, we use the matrix type defined in the PHAT library [3] and modify their reduction algorithm to also construct the chains that "fill" in the topological noise.

5 Experimental Results

The data used in our experiments comes from a large data set of CT scans of maize roots; data was processed according to the Shovelomics protocol [34] at Danforth Plant Science Center, and washed and imaged using a North Star Imaging (NSI, Rogers, MN) X5000 using NSI's efX-DR software. The end result is a 3d volume at 109μm voxel resolution. Our code is publicly available at `https://github.com/davidletscher/VHS/`.

These input scans are then segmented into the core, shape and neighborhood using the same 4 parameters for all scans. These are intensity thresholds (t_C, t_S, t_N) for each of the three regions plus and additional radius value (r). The shape, S, will consist of all voxels with intensity at least t_S. The neighborhood, N, will be the union of all voxels with intensity at least t_N unioned with a neighborhood of S of radius r. Finally, the core, C, will be the union of all voxels with intensity at least t_C and voxels who have a ball of radius r contained in S. This "blurring" of radius r reduces the impact of salt-and-pepper noise and topological noise in the spatial domain. Ideally we would use two-dimension persistence in threshold and radius to detect and repair noise in both the intensity and spatial domains, but such an approach is not computationally feasible. Our methods, in essence, allow us to work on a particular diagonal of the two-dimensional persistence filtration.

To obtain skeletons for these shapes, we use a the combination of methods in VoxelCore [37] and Erosion Thickness [38] which are general purpose methods that can computer a topologically correct one dimensional skeleton for any space homotopy equivalent to a graph. If perfectly scanned and segmented, corn roots will of course always be simply connected; however, errors in the scanning and reconstruction result in artificial loops and disconnected components. Ideally, the root shape is very close to that of a thickened tree, where the thickness monotonically decreases from the stem to the finer grained roots. However, an additional complexity to this data set is that the interior structure of the stem and lateral roots is not solid, as there are several voids in the interior of these roots. In practice, some of these voids are broken open due to damage of the roots in the harvesting, cleaning and drying process, which in turn motivates the necessity of the pseudo-void simplification.

Table 1: Experimental results: "Optimal Complexity" here is the sum of the Betti numbers of S. The three methods compared here heuristic solution to the homological simplification problem [7], and the techniques of this paper with and without the prior repair of pseudo-voids.

	Input Data Characteristics			Experimental Method		
	Number of Voxels ($\cdot 10^3$)	Initial Complexity	Optimal Complexity	Homological Simplification Time (s)	Without Pseudo-void Removal Time (s)	With Pseudo-void Removal Time (s)
`sw_08`	73	1,119	28	533	1,358	1,235
`sw_00`	83	990	57	617	961	895
`fl_01`	150	3,948	141	1,655	1,415	1,323
`321_3`	157	2,154	133	8,926	9,807	9,761
`314_4`	325	1,835	85	3,229	4,320	3,180
`fl_04`	418	13,477	254	16,525	18,237	18,111

Table 1 shows the runtime on six different corn roots at different stages of development, where optimal complexity here simply refers to the sum of Betti numbers of $H(C \hookrightarrow N)$. It is worth noting that as the size of the root systems grows, the number of topological errors increases quickly. Even in a perfect solution to the homological simplification problem all of these errors cannot be removed, as some are indeed present in the physical scans due to damage and dirt. As a result, the simplification will not be simply connected and generally will still contain several hundred topological errors. Typically, most of these appear as connectivity issues, although there are at most a few dozen loops. For comparison, we look at three different run times: a purely additive heuristic solution to the homological simplification problem [7], our methods without prior removal of pseudo-void, and our method with a preprocessing step to remove pseudo-voids. Our runtimes were on average 29% slower than the purely additive model. On average, the initial pass to remove pseudo-void reduced the runtime by 8%, as pseudo-void removal is a relatively fast process (at least compared to our cut and fill generation and testing) that removes a significant percentage of the topological errors. It is also worth noting, that in this set of experiments all of the simplifications were made on the first pass; however a second pass was needed to check for additional candidate simplifications.

All of our experiments were run on a server with 256 GB RAM and dual Xeon processors with ten cores each running at 2.2 GHz. For the experimental results, we focused on on corn roots at different stages of their growth. All timing results are the average over 10 runs.

Beyond run time, we evaluate the quality of our simplifications both in terms of topological accuracy in Section 5.1, and in terms of utility to skeleton analysis in Section 5.2, demonstrated skeletons from two common voxelized skeletonization packages. See Figure 2 for a visual of the improvement.

Table 2: The complexity of various methods. From left to right: unsimplified, theoretical optimum, homological simplification [7], and our techniques without and with the removal of pseudo-voids. Note that our method yields worse topological simplifications, when looking strictly in terms of Betti numbers; however, the geometric quality of the resulting skeletons is much higher, as shown later in Tables 4 and 5.

	Initial	Optimal	Homological Simplification	Without Pseudo-void Removal	With Pseudo-void Removal
`sw_00`	990	57	57	107	98
`314_4`	1,835	85	86	152	138
`321_3`	2,154	133	135	161	158
`fl_04`	13,477	254	254	307	298
% errors repaired on average			99.96%	97.26%	97.75%

5.1 Quality of simplifications

Our primary measure for the topological quality of the simplification is the sum of the Betti numbers. Table 2 shows a summary of results for prior work and the techniques introduced in this paper. The heuristic simplification based on the homological simplification problem [7] were most effective in reducing topological noise. These methods were purely additive and we will see the quality of the skeletons obtained using this simplification is not as high. It is unclear why the additive-only methods were able to outperform methods that could both add and remove portions of the shape to reduce the topological noise. For our methods, the removal of pseudo-voids removed an additional 17% of the topological errors.

Parameters to our software include the size of boxes to use to find candidate simplifications and whether the boxes overlap or are disjoint. Table 3 show the effects of the parameters on the quality of simplification and computation time. Overlapping boxes of larger sizes provided the most robust simplification, but using larger sized boxes that are disjoint boxes resulted in the fastest run time. Overall, making the boxes disjoint decreased the quality by 26% and decreased the run-times by 52%; using the larger sized boxes increased the quality by 16% while decreasing the run-times by 5%. For all of the other comparisons, we utilize a box size of 200 with overlaps to obtain the highest quality skeleton. It is not surprising that larger, overlapping boxes applied the most simplifications, as it detects and, hopefully, repairs larger features and those that span multiple regions.

5.2 Quality of Skeletons

As discussed previously, even when homological simplification is perfect, artifacts in the reconstruction can have drastic impact on later skeleton quality, when it is computed based on the simplified space. See Figure 9, where pseudo-voids can be seen in the left where the root was partially shredded along the stalk. Visually, it is clear that skeletons are much simpler after removing pseudo-voids

Table 3: The effects of box size and overlap of boxes on quality of result and run-time. The highest quality is obtained with overlapping boxes that are larger. The fastest is larger disjoint boxes. All of these tests utilized pseudo-void removal.

	Parameters	Initial Complexity	Optimal Complexity	Complexity	Time (s)
314_4	Box size = 50 overlapping	1,835	85	184	3,122
314_4	Box size = 50 disjoint	1,835	85	153	2,673
314_4	Box size = 200 overlapping	1,835	85	137	4,819
314_4	Box size = 200 disjoint	1,835	85	150	2,407
321_3	Box size = 50 overlapping	1,881	133	169	15,839
321_3	Box size = 50 disjoint	1,881	133	314	4,001
321_3	Box size = 200 overlapping	1,881	133	158	9,761
321_3	Box size = 200 disjoint	1,881	133	264	3,019

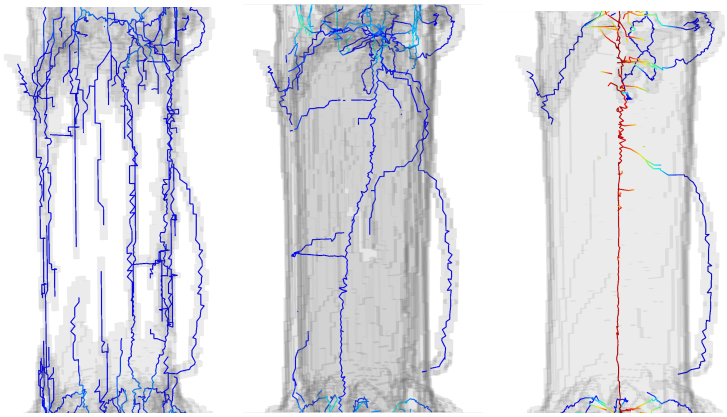

Figure 9: Three closeup views of the same corn root skeleton as shown in Figure 8, computed using [38, 37]; color indicates the distance from the skeleton to the shape boundary (larger values in red). Left: A skeleton after simplification using the purely-additive techniques of [7]. Middle: A skeleton after simplification, without pseudo-void removal. Right: A skeleton computed after both pseudo-void removal and topological simplification.

and topological noise. Even after simplification, the middle figure shows there is still a pseudo-void but with much less complexity, although the use of both cut and fills resulted in a more appealing solution. However, with pseudo-voids filled in the rightmost image, note that the skeleton passes down the center of the stalk, which more accurately reflects the ground truth.

To quantify this improvement in the quality of the skeletons we will use two different measures, the topological accuracy of the skeleton and how much a thickened skeleton covers the original shape. For topological accuracy in skeletons, we will again use Betti numbers, and sum the first two Betti numbers not measure the number of components and the number of loops in the skeleton. Our second quality measure is the percentage of the shape covered by the thickened skeleton, which we refer to as coverage. The justification for this measure is a bit more complex, and is based on the assumption that the ground truth is that our shapes are "thickened" graphs, or more specifically, that the graph is the medial axis of the shape. In such a case, there is a radius function on the graph and the shape is a union of balls of that radius centered on the graph. We will calculate the radius of a point on the graph as its distance to the boundary of the simplified shape. If we take the union of all of these balls, we obtain the thickened skeleton. In the ideal model this would be the entire shape, although we if the shape has regions that are more plate-like then a lower percentage of the shape would be in the thickened skeleton. (Note that this is particularly tailored to roots, which do not generally have large flat "plate-like" regions

Table 4 shows these two quality measures for three different corn roots. We examined the unsimplified shape, the additive only process for approximating a solution to the homological simplification problem [7], and our methods implemented in the VHS package both with and without pseudo-void removal. For each, we used skeletonization via the combination of VoxelCore [38] and Erosion Thickness [37] (VC+ET). Note that some topological information might be lost during skeletonization; while those methods guarantee homotopy equivalence on shapes that are thickened graphs, our roots here may have remaining 2-dimensional structure. This in part accounts for the drop in complexity from the shape to the skeleton from Table 2 to Table 5, as our shapes might

Table 4: The quality of skeleton measures for the unsimplified shapes, homological simplification [7], and our techniques without and with the removal of pseudo-voids. Each of the two methods were skeletonized using Voxelcore [38] and erosion thickness [37].

	Measure	Unsimplified Shape	Homological Simplification	Without Pseudo-void Removal	With Pseudo-void Removal
sw_00	$\beta_0 + \beta_1$	105	21	39	21
	Coverage	45%	51%	82%	96%
321_3	$\beta_0 + \beta_1$	1,191	51	69	59
	Coverage	28%	38%	80%	89%
314_4	$\beta_0 + \beta_1$	425	107	303	81
	Coverage	69%	78%	78%	92%

Table 5: A Comparison with TopoRoot.

	Measure	Unsimplified Shape	Homological Simplification	TopoRoot	VHS
321_3	$\beta_0 + \beta_1$ of shape	1,191	51	396	59
	$\beta_0 + \beta_1$ of skeleton	1,191	51	1	59
	Coverage	28%	38%	49%	89%
314_4	$\beta_0 + \beta_1$ of shape	425	107	601	81
	$\beta_0 + \beta_1$ of skeleton	425	107	1	81
	Coverage	69%	78%	53%	92%

have H_2 errors present but our skeletons will not.

We next compare to the other method utilized in TopoRoot [41], which reduces the cuts and fills to a graph problem and utilizes Steiner tree solvers to compute a heuristic solution [40]. We test this method on two of our roots, as seen in Table 5 and Figure 10. The other method is considerably faster than VHS, as expected, since they are building upon well developed fast Steiner tree solvers; in contract, the persistence calculations are much slower. Moreover, since TopoRoot is based on Steiner trees, it will always return a single tree, where as VHS does not perfectly simplify the shape. In practice, the TopoRoot method discards components that cannot be connected to the main tree; VHS's approach requires keeping all of the core, in contrast, which means the result is not a perfect tree. These small portions of the root can be seen in VHS's output but not in TopoRoot's in Figure 10; see near the base of the root, where isolated root pieces are floating in the VHS image. Elimination of these fragments in the TopoRoot image is what causes the significant reduction in coverage.

Our main observation from this data is that that more robust topological simplification methods significantly improved the geometric quality of the corn root skeletons. In particular, when we combine both topological repairs and filling of pseudo-voids, the skeletal coverage average rises to well over 90%. This implies that the simplified shape is very close to being a thickened version

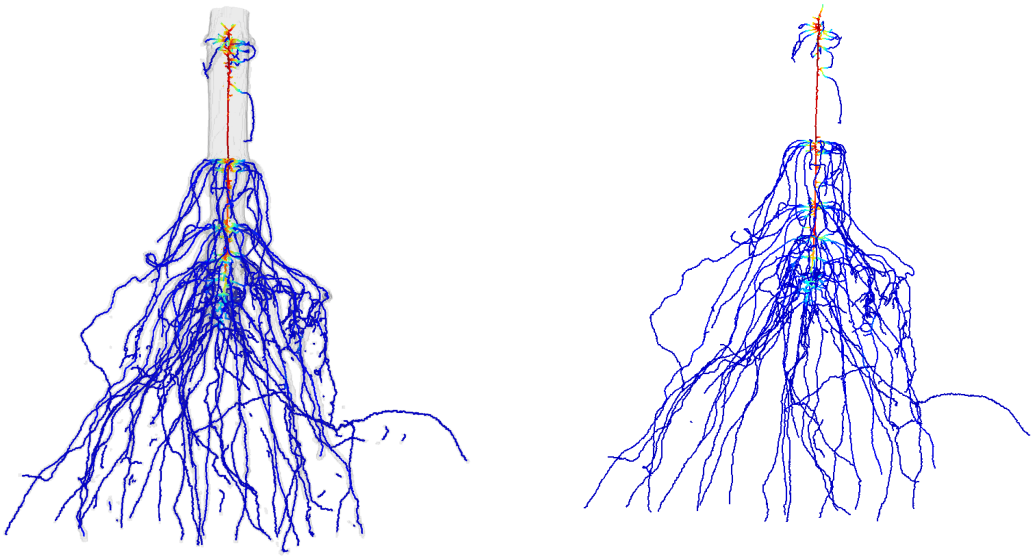

Figure 10: A skeleton obtained using (left) VHS and (right) TopoRoot. Note the extra pieces which are retained in the VHS-constructed skeleton, which result in a better skeleton coverage, while TopoRoot discards these fragments.

of the skeleton, at least for our root data set. This will allow our skeletons to be used in further analysis for biologically relevant traits. Unfortunately, there is an inevitable trade-off here, as the skeletons have slightly worse topological complexity in order to get better geometric centrality and coverage.

6 Conclusion and future work

There are several clear directions to pursue in order to speed up our topological repair process. While there has been some work on parallelizing persistent homology [2, 19], these are not immediately applicable in our setting because they do not construct the cuts and fills that our approach requires. However, in our approach, the candidate generation phase could potentially be parallelized with different processes working on separate windows. We note that validation of the cuts and fills takes much longer than the generation phase. If we were doing our computations using cubical complexes then much of the validation would be unnecessary due to the non-overlapping nature of cells in a cubical complex. However, the complexities of working with voxels force us to have to do this validation. If we can either construct "nicer" candidate simplifications or fast tests that would eliminate or reduce the need for validation, this would be a significant improvement.

It is also possible the tools from discrete Morse theory could be applied [8], as the Morse complex encodes quite a bit of geometry and may do better than simple persistence-based calculations. Using Morse theory could accelerate some computations, but it will add considerable complexity to the generation of candidates, as we need somehow to map back the obtained cycles to the original cubes. This remains an interesting avenue to explore in future work, however.

Finally, the topological repairs we made are deliberately small, but they might not be the most geometrically appealing. For example, two components could be connected by a very thin curve of voxels in our solution, where might be more geometrically appealing to connect them with a thicker curve that would match the rest of the shape better. This would extend the problem beyond just topological simplification to include more geometric aesthetics.

Beyond topological repair, we plan to conduct more extensive tests of roots systems to determine if these skeletons can successfully compute more localized and fine-grained traits in roots, such as thickness, branching angle, length, and tortuosity, and then search for genetic factors that control the traits. We also hypothesize that VHS will be useful for homological simplification of a variety of types of voxelized data. We have focused on skeletonization via VoxelCore [37] and Erosion Thickness [38] in our testing so far, as well as preliminary comparisons with Giaroots [33], but of course there are many other skeletonization algorithms to consider, some of which may be well suited to particular types of data. We conjecture that the appropriate combination of homological and pseudo-void simplification is likely to vary considerably, both with different data sets and different skeletonization packages.

Acknowledgements

This work was supported in part by the National Science Foundation, under grants DBI-1759836, EF-1921728, DBI-1759807, CCF-1614562, CCF-1907612, and CCF-2106672.

References

[1] Dominique Attali, Ulrich Bauer, Olivier Devillers, Marc Glisse, and André Lieutier. Homological Reconstruction and Simplification in $\mathbb{R}^3$. *Computational Geometry*, 48(8):606 – 621, 2015.

[2] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed Computation of Persistent Homology. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, page 31–38, USA, 2014. Society for Industrial and Applied Mathematics.

[3] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. PHAT – Persistent Homology Algorithms Toolbox. https://github.com/blazs/phat.

[4] Ulrich Bauer, Carsten Lange, and Max Wardetzky. Optimal topological simplification of discrete functions on surfaces. *Discrete & Computational Geometry*, 47(2):347–377, 2012.

[5] Amir Ben-Amram and Simon Yoffe. A Simple and Efficient Union-Find-Delete Algorithm. *Theoretical Computer Science*, 412:487–492, 02 2011.

[6] Afra J. Carlsson, Gunnar Zomorodian. The Theory of Multidimensional Persistence. *Discrete & Computational Geometry*, 42:71—93, 2009.

[7] Erin W. Chambers, Tao Ju, David Letscher, Mao Li, Christopher N. Topp, and Yajie Yan. Some Heuristics for the Homological Simplification Problem. In Stephane Durocher and Shahin Kamali, editors, *Proceedings of the* 30^{th} *Canadian Conference on Computational Geometry, CCCG 2018, August 8-10, 2018, University of Manitoba, Winnipeg, Manitoba, Canada*, pages 353–359, 2018.

[8] 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, March 2015.

[9] Tamal K. Dey, Kuiyu Li, and Jian Sun. On Computing Handle and Tunnel Loops. In *Cyberworlds, 2007. CW '07. International Conference on*, pages 357–366, Oct 2007.

[10] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. AMS Press, 2009.

[11] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological Persistence and Simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 454–463. IEEE, 2000.

[12] Herbert Edelsbrunner, Dmitriy Morozov, and Valerio Pascucci. Persistence-Sensitive Simplification Functions on 2-Manifolds. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 127–134. ACM, 2006.

[13] Wei Gao, Steffen Schlüter, Sebastian R. G. A. Blaser, Jianbo Shen, and Doris Vetterlein. A shape-based method for automatic and rapid segmentation of roots in soil from x-ray computed tomography images: Rootine. *Plant and Soil*, 441(1-2):643–655, June 2019.

[14] Stefan Gerth, Joelle Claußen, Anja Eggert, Norbert Wörlein, Michael Waininger, Thomas Wittenberg, and Norman Uhlmann. Semiautomated 3d root segmentation and evaluation based on x-ray CT imagery. *Plant Phenomics*, 2021:1–13, February 2021.

[15] Hannah Schreiber, David Letscher. VHS: a package for homological simplification of voxelized plant root data, 2020. `https://git.cs.slu.edu/public-repositories/shape-simplification-software/-/tree/master/VHS_v2.0`.

[16] Iko T. Koevoets, Jan Henk Venema, J. Theo. M. Elzenga, and Christa Testerink. Roots withstanding their environment: Exploiting root system architecture responses to abiotic stress to improve crop tolerance. *Frontiers in Plant Science*, 07, August 2016.

[17] Suxing Liu, Carlos Sherard Barrow, Meredith Hanlon, Jonathan P. Lynch, and Alexander Bucksch. DIRT/3d: 3d root phenotyping for field-grown maize (izea mays/i). *Plant Physiology*, 187(2):739–757, July 2021.

[18] Stefan Mairhofer, Susan Zappala, Saoirse R. Tracy, Craig Sturrock, Malcolm Bennett, Sacha J. Mooney, and Tony Pridmore. RooTrak: Automated recovery of three-dimensional plant root architecture in soil from x-ray microcomputed tomography images using visual tracking . *Plant Physiology*, 158(2):561–569, December 2011.

[19] Rodrigo Mendoza-Smith and Jared Tanner. Parallel Multi-Scale Reduction of Persistent Homology Filtrations. *arXiv: Algebraic Topology*, 2017.

[20] Dmitriy Morozov. *Homological illusions of persistence and stability*. Duke University, 2008.

[21] James R. Munkres. *Elements of algebraic topology*. CRC Press, 2018.

[22] Steve Y. Oudot. *Persistence Theory: From Quiver Representations to Data Analysis*, volume 209 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2015.

[23] J.S. Perret, M.E. Al-Belushi, and M. Deadman. Non-destructive visualization and quantification of roots using computed tomography. *Soil Biology and Biochemistry*, 39(2):391–399, February 2007.

[24] M.A Piñeros, B.G. Larson, J.E.Shaff, D.J. Schneider, A.X. Falcão, L.Yuan, R.T. Clark, E.J. Craft, T.W. Davis, P.L. Pradier, N.M. Shaw, I. Assaranurak, S.R. McCouch, C. Sturrock, M. Bennett, and L.V. Kochian. Evolving technologies for growing, imaging and analyzing 3d root system architecture of crop plants. *Journal of Integrative Plant Biology*, 58(3):230–41, March 2016.

[25] V Robins, P J Wood, and A 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, August 2011.

[26] Hannah Schreiber. *Algorithmic Aspects in standard and non-standard Persistent Homology*. PhD thesis, Graz University of Technology, 2019.

[27] Hannes Schulz, J. Postma amd D. V. Dusschoten, H. Scharr, and Sven Behnke. 3d reconstruction of plant roots from mri images. In *Proceedings of the International Conference on Computer Vision Theory and Applications*. SciTePress - Science and and Technology Publications, 2012.

[28] Mon-Ray Shao, Ni Jiang, Mao Li, Anne Howard, Kevin Lehner, Jack L. Mullen, Shayla L. Gunn, John K. McKay, and Christopher N Topp. Complementary phenotyping of maize root architecture by root pulling force and x-ray computed tomography. March 2021.

[29] Olga Symonova, Christopher N. Topp, and Herbert Edelsbrunner. DynamicRoots: A software platform for the reconstruction and analysis of growing plant roots. *PLOS ONE*, 10(6):e0127657, June 2015.

[30] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*, volume 35, pages 573–597. Wiley Online Library, 2016.

[31] 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.

[32] Julien Tierny and Valerio Pascucci. Generalized Topological Simplification of Scalar Fields on Surfaces. *IEEE transactions on visualization and computer graphics*, 18(12):2005–2013, 2012.

[33] C. N. Topp, A. S. Iyer-Pascuzzi, J. T. Anderson, C.-R. Lee, P. R. Zurek, O. Symonova, Y. Zheng, A. Bucksch, Y. Mileyko, T. Galkovskyi, B. T. Moore, J. Harer, H. Edelsbrunner, T. Mitchell-Olds, J. S. Weitz, and P. N. Benfey. 3d phenotyping and quantitative trait locus mapping identify core regions of the rice genome controlling root architecture. *Proceedings of the National Academy of Sciences*, 110(18):E1695–E1704, April 2013.

[34] Samuel Trachsel, Shawn M. Kaeppler, Kathleen M. Brown, and Jonathan P. Lynch. Shovelomics: high throughput phenotyping of maize (zea mays l.) root architecture in the field. *Plant and Soil*, 341(1-2):75–87, November 2010.

[35] Hubert Wagner, Chao Chen, and Erald Vuçini. *Efficient Computation of Persistent Homology for Cubical Data*, pages 91–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Zoë Wood, Hugues Hoppe, Mathieu Desbrun, and Peter Schröder. Removing Excess Topology from Isosurfaces. *ACM Transactions on Graphics (TOG)*, 23(2):190–208, 2004.

[37] Yajie Yan, David Letscher, and Tao Ju. Voxel cores: efficient, robust, and provably good approximation of 3d medial axes. *ACM Trans. Graph.*, 37(4):44:1–44:13, 2018.

[38] Yajie Yan, Kyle Sykes, Erin Chambers, David Letscher, and Tao Ju. Erosion thickness on medial axes of 3d shapes. *ACM Trans. Graph.*, 35(4):1–12, 2016.

[39] Dan Zeng, Erin Chambers, David Letscher, and Tao Ju. To cut or to fill. *ACM Transactions on Graphics*, 39(6):1–18, November 2020.

[40] Dan Zeng, Erin W. Chambers, David Letscher, and Tao Ju. To cut or to fill: a global optimization approach to topological simplification. *ACM Trans. Graph.*, 39(6):201:1–201:18, 2020.

[41] Dan Zeng, Mao Li, Ni Jiang, Yiwen Ju, Hannah Schreiber, Erin Chambers, David Letscher, Tao Ju, and Christopher N. Topp. TopoRoot: a method for computing hierarchy and fine-grained traits of maize roots from 3d imaging. *Plant Methods*, 17(1), December 2021.