High-Quality Progressive Alignment of Large 3D Microscopy Data

Aniketh Venkat*
SCI Institute, University of Utah
Duong Hoang
SCI Institute, University of Utah
Attila Gyulassy
SCI Institute, University of Utah
Peer-Timo Bremer
Lawrence Livermore National Labs
Frederick Federer
Moran Eye Institute, University of Utah
Alessandra Angelucci
Moran Eye Institute, University of Utah
Valerio Pascucci
SCI Institute, University of Utah

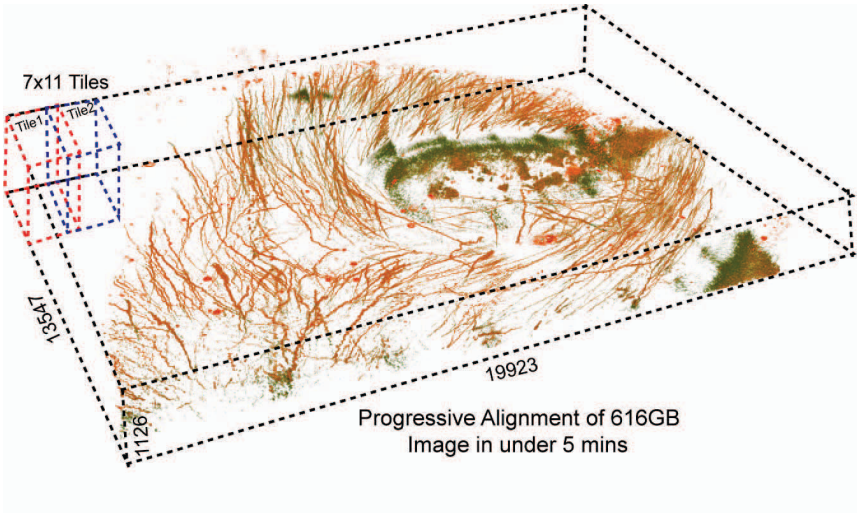

Figure 1: Our progressive alignment approach working on a 3D dataset comprising a grid of 7×11 tiles, each of dimensions $2048 \times 2048 \times 1000$. The total size of the dataset is 616 GB. All the tiles after alignment and stitching. The whole alignment process took 4.97 minutes (running on 72 parallel threads). The Peak RAM usage on a 144 core CPU machine was 3 GB.

Abstract

Large-scale three-dimensional (3D) microscopy acquisitions frequently create terabytes of image data at high resolution and magnification. Imaging large specimens at high magnifications requires acquiring 3D overlapping image stacks as tiles arranged on a two-dimensional (2D) grid that must subsequently be aligned and fused into a single 3D volume. Due to their sheer size, aligning many overlapping gigabyte-sized 3D tiles in parallel and at full resolution is memory intensive and often I/O bound. Current techniques trade accuracy for scalability, perform alignment on subsampled images, and require additional postprocess algorithms to refine the alignment quality, usually with high computational requirements. One common solution to the memory problem is to subdivide the overlap region into smaller chunks (sub-blocks) and align the sub-block pairs in parallel, choosing the pair with the most reliable alignment to determine the global transformation. Yet aligning all sub-block pairs at full resolution remains computationally expensive. The key to quickly developing a fast, high-quality, low-memory solution is to identify a single or a small set of sub-blocks that give good alignment at full resolution without touching all the overlapping data. In this paper, we present a new iterative approach that leverages coarse resolution alignments to progressively refine and align only the promising candidates at finer resolutions, thereby aligning only a small user-defined number of sub-blocks at full resolution to determine the lowest error transformation between pairwise overlapping tiles. Our progressive approach is 2.6x faster than the state of the art, requires less than 450MB of peak RAM (per parallel thread), and offers a higher quality alignment without the need for additional postprocessing refinement steps to correct for alignment errors.

Keywords: Alignment, Stitching, Normalized Cross-Correlation, Progressive Computations, Coarse-to-Fine, Microscopy, Terascale

1 Introduction

Recent advances in microscopy have enabled researchers to acquire large amounts of data at high resolution and magnification. Consequently, the samples are often too large to fit into the field of view of a microscope. Computer-controlled micropositioning stages acquire a single tile through the depth of the tissue (as a stack of 2D images) at a given X, Y coordinate, the scan then moves to the next X, Y coordinate, maintaining a fixed overlap with adjacent tiles, until the entire region of interest has been imaged. As the microscope moves from one position to the next, scanning one field of view at a time, a range of movements often causes the data to be misaligned. Reconstructing a 3D image of the specimen requires precise alignment of many overlapping 3D tiles. We target the use case where overlapping 3D tiles need a fine-scale alignment, for instance in multi-photon, lightsheet, or single slab confocal microscopy.

One of the key challenges in efficiently aligning terabyte-sized datasets is the I/O bottleneck between disk and memory. Existing alignment approaches often circumvent the problem by computing coarse alignments using downsampled tiles, and optionally refining the results using full-resolution tiles. However, a single coarse-scale alignment, followed by a few full-resolution refinements in

*e-mail: aniketh.venkat@utah.edu

2022 IEEE 12th Symposium on Large Data Analysis and Visualization (LDAV) | 978-1-6654-9156-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/LDAV57265.2022.9966406

the neighborhood of the coarse-scale estimate does not always find the best global transformation. Yet, in reality, very few techniques have explored a more rigorous approach to align terascale images efficiently using coarse-scale alignments. The reason is twofold: first, the coarse alignments can be unreliable if the resolution chosen is too coarse; if a mistake is made at a coarse scale, it can be impossible for the fine-scale alignment to succeed. Second, at full resolution, although the search space is reduced by leveraging coarse alignments, there can still be significant amounts of data to be fetched for fine-scale alignment to succeed. Current state-of-the-art tools thus often take several hours to days on a moderate desktop and a few hours on a large server to align and stitch terabyte-sized images at full resolution. A trade-off between precision and performance is generally required to improve the compute time.

Our approach is based on the crucial observation that if we divide the overlapping regions into sub-blocks, few sub-blocks are required at full resolution to obtain accurate alignments, and the majority of the sub-blocks can safely be pruned after examining their inexpensive coarse-scale alignments. We address the above challenges by performing progressive computations, starting with a coarse resolution, and refining as needed only the sub-blocks that are likely to provide good alignment at full resolution. Realizing this approach requires us to solve a number of fundamental problems: how to choose the starting resolution level, how to reliably eliminate the sub-blocks that likely do not contain enough information for good alignment, how to efficiently refine a sub-block's resolution, and how to efficiently perform low-level pairwise alignment of sub-blocks.

Contributions. To address these challenges, our specific technical contributions are:

- A heuristic to determine the minimum resolution required to make informed decisions on what sub-blocks to keep/discard based on the percentage of sub-blocks that provide consistent alignments between successive resolutions and the signal-to-noise ratio
- A new hybrid method to estimate pairwise alignment using 3D NCC at coarse resolutions and a fast maximum intensity projection based 2D approximation at finer resolutions
- A probabilistic approach to decide which sub-blocks to use for alignment at a higher resolution based on a two-stage stratified weighted random sampling
- The above contributions are made possible through a novel I/O scheme utilizing efficient progressive refinement of sub-block resolution without re-fetching data from disk.

Together, our contributions amount to a novel, fast, and an iterative approach that leverages coarse-scale computations to progressively guide fine-scale computations to fetch higher resolution data where it matters most for alignment refinement, guaranteeing a high-quality result with low computational requirements. Unlike previous approaches, our method is progressive and adaptive, since it utilizes multiple resolution levels as needed, and sub-blocks can be dropped at different resolution levels depending on whether they are useful at the next finer level. This method gives users a flexible choice of Quality vs. Speed: when a fast solution is desired, our approach seamlessly skips multiple refinements; to maximize quality it uses all the resolution levels. We demonstrate the efficacy of our approach through extensive experiments, which show orders of magnitude speed-up over the state-of-the-art methods while requiring less memory and offering high-quality alignments.

2 Background and Related Work

Large Microscopy Acquisitions. The primate brain contains billions of neurons connected in specific ways to form complex circuits. Brain function emerges from the activity of these circuits, and long-range connections between different brain areas dictate the flow of information. A major goal of modern neuroscience is to obtain a wiring diagram of neural circuits across the entire brain at different scales, a field called connectomics. While initial efforts to create a connectome have focused on rodent brains [3, 7], mapping the non-human primate (NHP) brain is rapidly becoming a feasible goal. To effectively resolve and visualize individual fluorescently labeled axons (which frequently cross and overlap each other through the NHP cortex), a minimum resolution of 20-40x magnification in the x-y plane combined with a z-axis step size of 0.25-1.0 microns between imaging planes is necessary. At this magnification and z-resolution, researchers can unambiguously identify continuous neuronal projections and the axonal and dendritic protrusions along them, which identify synaptic points of contact with other cells. A consequence of this high magnification is that the samples are often too large to fit into the field of view of a microscope. Therefore, computer-controlled micropositioning stages are used to acquire images as overlapping tiles arranged according to a 2D grid to image an entire specimen. One of the first challenges for a neuroscientist is to automatically align and reconstruct a 3D image of the specimen for visualization and analysis.

Image Alignment Methods. Image alignment methods are broadly classified into intensity-based [1, 28, 32, 33] and feature-correspondence-based algorithms [2, 8, 20, 25, 29, 37, 39]. Intensity-based approaches derive a cost function using the image intensity values and solve an optimization problem to align images. In contrast, feature-based methods extract image features and feature correspondences to fit a parametric model using ordinary least squares. For images with overlap, intensity-based methods are more suitable due to their simplicity and effectiveness. An image dissimilarity/similarity metric is minimized/maximized to find the unknown coordinate transformation that best matches the overlapping images. In pattern matching, this problem is extensively studied and is commonly referred to as template matching [6, 15, 16, 23, 36]. A small patch from the source image (template) is matched to a set of nonoverlapping candidate patches in the target image by computing a similarity score for each pair of matches. The pair with the highest score determines the final transformation. Recent research has extended the class of solutions to transformations with subpixel accuracy and rigid [26] and affine transformations [21]. For microscopy images recovering 3D translations is typically sufficient to align the data.

Correlation in 3D Images. To retrieve 3D translation from overlapping images, the sum of squared differences in the intensity values is minimized or equivalently the cross-correlation is maximized using PC [12] or NCC [23]. A full exhaustive search subdivides the overlap region into small 3D blocks, and repeatedly applies the correlation algorithm to each pair of blocks to determine the best match, hence the 3D translation. PC is efficiently calculated in the transform domain as the inverse Fourier transform of the normalized cross-spectrum between the two images. It is fast, resilient to noise, and often used at coarse resolutions to find rough estimates for the translation. However, PC fails when the local image variance is not constant. NCC overcomes this challenge by normalizing the image variance in both the source and the target 3D blocks. Nonetheless, the normalization step is expensive since it is required to compute the local image variance using spatial domain sliding windows. Precomputing image integrals using summed area tables [9] often speeds up the normalization. Although this algorithm is easy to implement and fast at coarse resolutions, the main obstacle to directly using it to align terabyte-sized 3D images at full resolution is the computational cost to calculate the similarity metric for all pairs of blocks and the cost of frequently moving large amounts of data from disk (sometimes stored out of core) to main memory.

Related Alignment Tools. Prior solutions to align large-scale microscopy data broadly fall into three groups: divide and conquer-

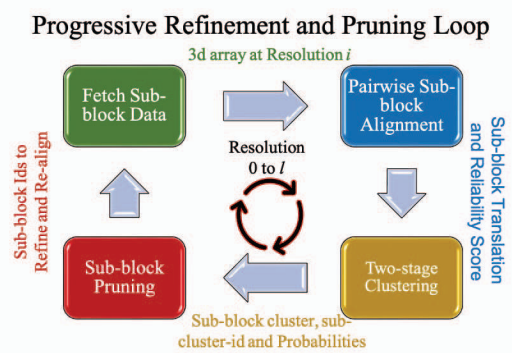

Figure 2: Overview of the progressive refinement and pruning algorithm. For each tile pair, the overlap region is sub-divided into sub-blocks, and for each corresponding sub-block pair, we repeat the alignment, clustering, pruning, and refinement loop, starting from the coarsest resolution (0), until the full resolution (l).

based solutions, interest point-based methods, and multiresolution techniques. Divide-and-conquer solutions subdivide the overlap region into small blocks; and compute the correlation for each block, potentially using an approximation to speed up the alignment. *TeraStitcher* [5] has extensively studied this class of solution. *TeraStitcher* [5] subdivided the overlap region into multiple z-slabs and computed the alignment using NCC and 2D projections of the z-slabs. This approach provides fairly accurate results for images with axis-aligned features and smooth gradients. However, for the images we tested, we noticed large alignment errors. Additionally, this approach touches the entire overlap region at full resolution and computes 2D NCC using sliding windows, both of which significantly impact runtime. A recent extension to this framework [4] addresses some of the limitations and accelerates the alignment computation using GPUs. Our approach uses 3D NCC at coarse resolutions, and inspired by this approach, we switch to a fast 2D approximation as blocks are refined.

Interest-point-based methods identify regions with high information density and compute correlations only around a small window centered on the interest points. *XuvTools* [10] aligns pairwise images at full resolution by computing NCC in regions with a high gradient magnitude. By restricting the combined volume of the regions to a small portion of the overlap, this approach yields fast high-quality results. However, this solution does not scale to terabyte-sized datasets, often running into out-of-memory issues. *BigStitcher* [19] circumvents the memory problem by working on downsampled images using PC with subpixel precision to align large overlapping tiles. Any initial alignment errors are corrected by utilizing a variation of the iterative closest point algorithm [31] in the postprocess. For optimal performance, this approach requires converting the raw data to HDF5 — a multiresolution, block data format. HDF5 supports multiresolution using mipmaps, leading to data duplication. Although this implementation supports full-resolution alignment, it is significantly slower. We avoid the data duplication problem by leveraging the IDX data format [22], which rearranges the data samples according to hierarchical space-filling curves.

Yu *et al.* [38] proposed *iStitch*, a two-step multiresolution approach to speed up the alignment computation. In step one, interest points are identified using PC and NCC on 125x downsampled images to determine a rough estimate for the translation. In step two, the final transformation is found at full resolution by computing PC and NCC within a small window centered around the best location estimate from the coarse resolution. Even though this approach succeeds in improving the runtime, using a single fixed coarse resolution to find rough estimates may produce alignment mistakes from which recovery is not possible. In contrast, our approach uses all

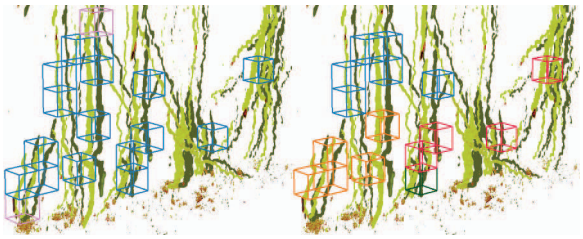

Figure 3: A partition of the sub-block transformation space into clusters and subclusters using two-stage stratification. Here, stage one partitions the transformation space into two clusters (blue and light green), and eliminates the noisy sub-blocks (light pink). Stage two further partitions the remaining cluster into four subclusters (blue, orange, dark pink, and dark green). Note, we only show the overlapping region between two tiles, rendered using contrasting color palettes (olive and seaweed green) to highlight the initial misalignment.

the resolution levels (computed on the fly), progressively reducing the number of computations at each refinement, concurrently improving the reliability of each estimate, and thereby guaranteeing a high-quality solution.

Performing progressive computations can significantly reduce the problem's complexity and provide a fast, low-cost solution even when working with terabyte-sized images. We use ideas from coarse-to-fine template matching [14, 24, 26, 34] to efficiently reduce the number of computations at full resolution. Coarse-to-fine template matching is based on efficiently examining a lower bounding function to rapidly eliminate mismatching candidates at coarse resolutions, thereby minimizing the number of expensive computations at full resolution. The lower bounding function is usually generated using a projection kernel [17], and with each refinement in resolution, the bound gets tighter, eliminating further candidates. This process is repeated until full resolution, where only a few remaining candidates are matched to find the best transformation. Ouyang *et al.* [27] compared state-of-the-art algorithms to compute the lower bounding functions in a unified framework. Our approach is motivated by this line of research and is similar in the following aspects: first, we leverage coarse-resolution alignments to carefully prune candidates that are unlikely to contribute to the full-resolution alignment, and, second, with each refinement in resolution, the accuracy of our solution improves, thereby guaranteeing a high-quality alignment at full resolution. However, we significantly differ by avoiding computing the lower bounding function since it requires fetching the entire overlap region at full resolution to compute the coarse-resolution template and candidates, which is expensive. Instead, we employ a probabilistic approach using two-stage stratified weighted random sampling to find the candidates to refine at the next higher resolution. Our solution is progressive and does not require fetching the entire overlap at full resolution to align pairwise volumes.

3 Approach

Formally, the alignment problem we aim to solve can be stated as follows. The input is a set of tiles, $\mathbf{T}$, that overlap in 3D. A tile is a 3D volume corresponding to the 2D stack of raw image data. The outputs are a set of 3D translations for each pair of overlapping tiles in $\mathbf{T}$. When two tiles $t_1, t_2 \in \mathbf{T}$ overlap, we define two sets of sub-blocks in the overlapping region, $\mathbf{S_1}$ and $\mathbf{S_2}$ ($\mathbf{S_i} \in t_i$). We denote the total number of sub-blocks using N. The spatial extent of the sub-blocks does not change throughout the alignment process, but the number of data samples they contain will change. Our progressive approach starts from the coarsest resolution level and computes alignment for each corresponding sub-block pair. The alignment computation (Sect. 3.1) assigns a 3D translation and a reliability score that is used to group the sub-blocks into

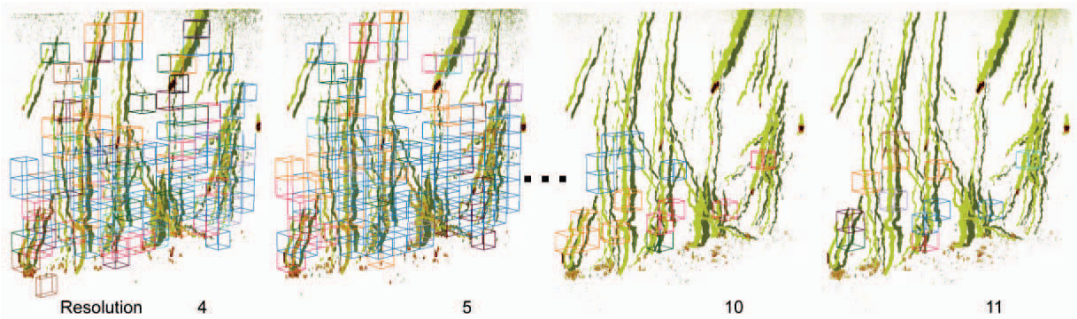

Figure 4: Progressive pruning and refinement of the sub-blocks. The volume rendering only shows the overlapping region between two tiles, rendered using contrasting color palettes (olive and seaweed green) to highlight the initial misalignment. Starting after prune-start resolution (far left), a two-stage stratification subdivides the sub-block transformation space into clusters and subclusters at each resolution. The sub-blocks refined at each resolution are colored based on the subcluster-id. At full resolution (far right), only the remaining sub-blocks are aligned to determine the final transformation.

clusters and subclusters (Sect. 3.2). A probability value is assigned for each sub-block using the cluster and the sub-cluster size, and the sub-blocks to refine at the next resolution level are selected using weighted random sampling (Sect. 3.3). With each refinement, the number of sub-blocks to align decreases exponentially, and the alignment, clustering, pruning, and refinement loop is repeated until full resolution. Fig. 2 illustrates the progressive pruning, and refinement loop. At the full resolution, only the remaining sub-blocks after the coarse-to-fine elimination process are aligned to compute the final transformation for the pair of tiles (Sect. 3.4). We leverage the IDX [22] multiresolution format to service region-at-resolution requests on the fly (Sect. 3.5). Finally, all tile-pair transformations are combined to construct an aligned image.

3.1 Alignment Computation

Given a pair of sub-blocks, the alignment computation finds the transformation that maximizes the normalized cross-correlation (NCC) between them. The correlation is computed in the Fourier domain, using FFTW [13] to compute FFT, and summed area tables are used to speed up the normalization [23]. Similarly to [5], we use two reliability measures: *NCC peak value* (the higher, the more reliable), and *NCC shape width* of the peak to assign a reliability score for a sub-block pair. The NCC shape width is the radius at which NCC value drops to 80% of the peak, with smaller values indicating higher reliability. The reliability score is the product of the NCC peak value and one minus mean NCC shape width, with the mean width normalized by a maximum tolerance constant, empirically set to 30.

We compute precise pairwise transformations using 3D NCC at coarse-resolution levels and switch to a fast 2D approximation based on maximum intensity projection (MIP) and 2D NCC at finer resolutions. Although the spatial extent of a sub-block remains the same throughout the alignment process, the number of samples contained in a sub-block increases with resolution. Therefore, as resolution increases, computing NCC in 3D becomes expensive. Replacing the expensive 3D computation with the 2D approximation results in significant speed-up, without a major drop in quality. Our implementation provides users the flexibility to choose the resolution level to switch from 3D NCC to the 2D approximation, thereby allowing the use of 3D NCC at all resolution levels when accuracy is critical. For the 2D approximation, we compute, as in [5], MIP along the axis directions and 2D NCC for each pair of MIPs, resulting in two translations per direction. In contrast to [5], we assign the translation value of the most reliable MIP to the corresponding directions, and for the remaining direction, the most reliable translation along this direction is assigned from the other two MIPs. We evaluate the advantage and the cost in terms of the alignment quality of switching to the 2D approximation at fine resolutions in Sect. 4. The alignment computation returns a transformation and a reliability score.

3.2 Clustering

Given the sub-block transformation and reliability score, we partition the sub-block transformation space into clusters and subclusters using two-stage stratification. First, we assign each reliable sub-block to a cluster. The clustering is done in the space of sub-block translations using DBSCAN [11]. DBSCAN groups high-dimensional vectors into dense connected regions using two parameters: minimum number of points to form a cluster (MINPOINTS) and the proximity of points to be considered as the same cluster (EPSDISTANCE). We use the Euclidean distance to measure proximity and set MINPOINTS to two and EPSDISTANCE to one to form the clusters. We denote the sub-blocks that do not belong to any cluster as non-informative blocks (noise), and these blocks will not be considered further. Since DBSCAN forms density-connected clusters, a cluster can span a large extent of the transformation space. Hence, the first stage of stratification usually results in a small number of clusters. To add more structure to the partitioning, we use the second stage of stratification. In the second stage of stratification, we split each cluster into a subcluster by grouping the sub-blocks with identical transformations into a subcluster. We set MINPOINTS and EPSDISTANCE to zero to form the subclusters. Fig. 3 illustrates the two-stage stratification. The partitioning of the sub-block transformation space into clusters and subclusters aids the early elimination of sub-blocks that are not useful at the next finer level.

3.3 Pruning and Local Refinement

We take a probabilistic approach to determine the sub-blocks to refine at the next resolution level, using weighted random sampling. For each cluster, we assign a probability value as the ratio of the number of sub-blocks in the cluster to the total number of sub-blocks in all the clusters, and for each resulting subcluster (within a cluster), we assign a probability value as the ratio of the number of sub-blocks in a subcluster to the total number of sub-blocks within its parent cluster. We order the sub-blocks (largest to smallest) within each subcluster based on the reliability score so that when a subcluster is sampled, the most reliable sub-block is chosen first. By choosing weighted random sampling to select clusters and subclusters, we avoid introducing bias into our sampling. We make a deterministic choice only when a cluster and a subcluster are sampled, i.e., we choose the most reliable block first to be refined. In contrast, sampling the clusters/subclusters directly by sorting them by size and/or reliability introduces bias. We note that at coarse resolutions, it is not possible to make a deterministic choice as to which cluster will align a higher percentage of the overlap at full resolution as the clusters are yet to fully develop. With refinement, clusters can continue, split or merge. Therefore, it not always true that the largest cluster/subcluster at resolution i continues to be the largest cluster/subcluster at resolution $i+1$.

We select the sub-blocks to refine for the next higher resolution

(r) by sampling one at a time: first, the clusters (using the cluster probabilities as weights) to select a cluster and then the subclusters of the selected cluster (using the subcluster probabilities as weights) to choose a sub-block. The number of sub-blocks that are refined at the next resolution (r) is found using the expression $N.(n_{fs}/N)^{(p-r)/p}$, where N is the total number of sub-blocks, n_{fs} is the number of sub-blocks to align at full resolution, and p is the minimum resolution to start the pruning process. The remaining sub-blocks are pruned at the current resolution and will not be considered further. We describe the heuristic used to choose the minimum resolution (p) in Sect. 3.6. Once an entire subcluster (or cluster) is exhausted, we recompute the weights for random sampling. Fig. 4 illustrates the pruning process.

3.4 Full Resolution and Final Transformation

At full resolution, given the remaining sub-blocks after coarse-to-fine pruning, we find the best transformation that aligns the tile pair. In most cases, aligning the remaining sub-blocks at full resolution and using the transformation corresponding to the most reliable sub-block is sufficient to align the tile pair. However, in some cases, we noticed that more than one translation potentially aligned a portion of the overlap (Fig. 12). In order to select the translation that best aligns the majority of the overlap, we take an alternate approach to assign the best transformation. We realign the remaining sub-blocks at full resolution and group the sub-blocks with unique transformation into clusters. The resulting clusters are sorted (highest to lowest) based on cluster size, breaking ties using the average reliability score of all the sub-blocks within a cluster. We discard clusters with an average reliability score of less than 20% as unreliable, and from the remaining clusters, we choose the most reliable sub-block from the largest cluster as the candidate block for the final transformation. Next, we select additional candidates from the remaining clusters whose relative transformation distance is less than or equal to five voxels (empirically chosen) to the candidate sub-block's transformation (since the transformation in our case is a 3D translation, the distance between transformations is measured using the Euclidean distance metric). The transformation of the sub-block closest to the centroid of the candidates' transformation is chosen as the final transformation between the pair of tiles. When the cluster size is zero, the transformation of the sub-block with the largest reliability score is chosen as the final transformation between the pair of tiles.

Global Tile Placement. Once the pairwise alignment for all tile pairs is completed, we perform a global optimization step to find the final tile positions. We take a similar approach to BigStitcher [19], to build point matches for each overlapping tile pair. For the moving tile, we transform the 8-point bounding box of its overlap using the computed pairwise transformation and build a point correspondence to the bounding box of the fixed tile's overlap. We use the iterative minimization scheme of Saalfeld *et al.* [30], and minimize the square displacements of the point matches to find the final tile positions.

3.5 Progressive Loading of Sub-Blocks

For efficient loading of low-resolution data from disk, the common row-major order of storing grid samples is undesirable, since samples at different resolution levels are stored and fetched together. Instead, we leverage the multiresolution IDX data format [22], which decomposes a grid of samples $\mathbf{G}$ into a series of nested subgrids $\mathbf{G}_0 \subset \mathbf{G}_1 \subset \cdots \subset \mathbf{G}_{l-1} = \mathbf{G}$. Starting from $\mathbf{G}$, each subgrid $\mathbf{G}_n$ is formed by collecting all even-indexed samples in $\mathbf{G}_{n+1}$ in one of the dimensions (x, y, or z) and discarding the rest of the samples, so that $\mathbf{G}_n$ contains half the number of samples as $\mathbf{G}_{n+1}$. The indexing scheme is such that if $\mathbf{G}_{n+1}$ has $2^x \times 2^y \times 2^z$ samples, they will be given indices $(0,0,0)$ to $(2^x-1, 2^y-1, 2^z-1)$, inclusively. $\mathbf{G}_n$ then consists of samples in $\mathbf{G}_{n+1}$ with indices of the form (i,j,k) where $i \in [0,2,4,\ldots,2^x-2]$, $0 \le j \le 2^y-1$, and $0 \le k \le 2^z-1$, assuming that the split dimension is x. Each subgrid $\mathbf{G}_n$ is called a *resolution level*: $\mathbf{G}_0$ is the coarsest resolution level and $\mathbf{G}_{l-1}$ is the finest; the whole domain is thus decomposed into l resolution levels.

On disk, the resolution levels are stored separately in units of *blocks* of 2^B samples each (the first few resolution levels are grouped together in the first block if necessary). The blocks form the unit of disk I/O, i.e., whenever a sample is requested, the entire block containing that sample is read from the disk. On each level, the individual samples are sorted using Morton codes [35] prior to blocking, so that each sample is conceptually a leaf of a k-d tree, with the root representing the whole domain (potentially padded to have power-of-two dimensions). This scheme allows for efficient region-of-interest queries: given two inputs – a resolution level L and a 3D extent E (an axis-aligned box in 3D) – the k-d tree can be traversed in logarithmic time to determine blocks that overlap with E at resolution levels no finer than L. These blocks are then fetched from disk, and relevant data samples are copied into an output memory buffer denoted as $B_{L,E}$, ready to be used for further processing tasks.

In our case, each extent E is the bounding box of a sub-block; we keep the same extents but progressively query for more data samples inside the extents (that is, increasing their resolution). Because the resolution levels are nested, if finer resolution samples are later needed for the same extent E, the buffer $B_{L,E}$ can be expanded to accommodate new samples at level $L+1$ to be fetched from disk, while retaining the existing samples. This progressive loading process avoids fetching $B_{L+1,E}$ entirely from disk, which would be much more expensive. The updated buffer, $B_{L+1,E}$, is a finer resolution representation of the 3D field contained in E. In our implementation, we use the HANA file reader [18], which provides progressive loading for IDX encoded data.

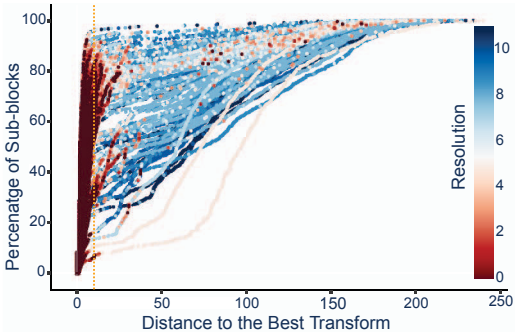

Figure 5: A visual validation of our pruning process. Each point in the figure represents a sub-block, and the color indicates the resolution at which the sub-block was discarded during our coarse-to-fine alignment computation. In addition, it shows the L2 error between the sub-block's full resolution transform and the baseline best transform. The sub-blocks to the left of the vertical line x = 10 are refined at resolutions $l-3$ or higher and their L2 error is less than 10 voxels.

3.6 Heuristic for Reliable Prune-Start Resolution

The success of a coarse-to-fine alignment approach is contingent on how the minimum resolution (p) to start pruning the sub-blocks is chosen. Choosing a fixed starting resolution for each dataset or at random can lead to large unforeseen alignment errors and slower execution times. A good starting resolution must be stable; it should contain a high percentage of blocks with sufficient detail to systematically identify the next set of blocks to refine. To identify a stable resolution with high percentage of informative blocks, we cluster sub-block transformations using DBSCAN, setting MINPOINTS to one percent of the total number of sub-blocks (N) and EPSDISTANCE

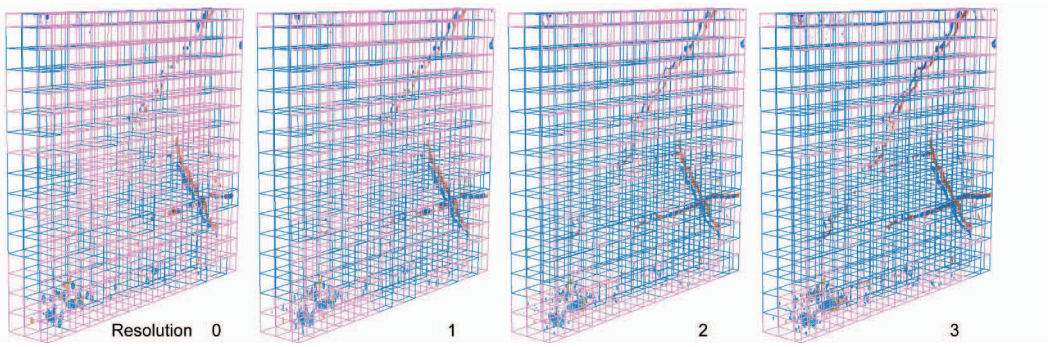

Figure 6: An example of how the prune-start resolution is determined. Here we show an overlapping region between two tiles, rendered at multiple coarse resolution levels. The tiles use contrasting color palettes (blue and red) to highlight the initial misalignment. The bounding boxes represent sub-blocks, with color that encodes the classification of the sub-block into signal or noise. Most sub-blocks at resolution levels 0 and 1 are noise (light pink bounding boxes). In contrast, most sub-blocks at resolution 2 are signal (blue bounding boxes). Between resolution levels 1 and 2, most sub-blocks change their classification from being signal to noise and vice versa; between resolutions 2 and 3, most sub-blocks retain their membership. At resolution level 3, there are enough self-consistent sub-blocks and a high signal-to-noise ratio to start eliminating the sub-blocks. This resolution is called the prune-start resolution.

to *one*, to group sub-blocks into clusters of size $\geq$ MINPOINTS. Setting MINPOINTS to one percent of the total number of sub-blocks minimizes the number of false positives. We refer to a sub-block that is not part of any cluster as *noise* and a sub-block with nonzero cluster-id as *signal*. To determine whether a resolution is stable or not, we check for self-consistency between successive resolutions.

We define a resolution to be stable when there is minimal change in sub-block membership between successive resolution levels. A sub-block can change its membership (i.e., from signal to noise or vice versa) at coarse resolutions for two reasons: first, it is typical for random blobs in the sub-blocks to correlate, leading to false positives (noise as signal), and second, insufficient resolution can lead to incomplete features that do not correlate, leading to false negatives (signal as noise). Usually, one refinement in resolution is sufficient to detect such unexpected correlations. We measure stability by counting the number of sub-blocks that are identified as signal or noise in successive resolutions. We label these sub-blocks as self-consistent. In addition, if the distance between the sub-block transformation at the two resolutions is less than or equal to a fixed distance threshold, we consider those sub-blocks to be self-consistent. The distance threshold accounts for isolated points in the transformation space that are incorrectly classified as noise. We typically set the distance threshold (d) to a small value (two voxels). We start pruning sub-blocks when the percentage of self-consistent blocks and the signal-to-noise ratio is above a user-defined threshold. Fig. 6 illustrates our approach in action.

For tiles with sparse tiny structures or a very low signal-to-noise ratio, we note that the percentage of self-consistent sub-blocks or signal-to-noise ratio can be below the user-threshold even at fine resolutions, which leads to aligning the entire overlap at all the resolutions levels. We avoid this scenario by allowing users to configure the default prune-start resolution. Once the default start resolution is reached, we start the pruning process (if it has still not started). Note, up to the prune-start resolution, all the sub-blocks are aligned in parallel for each successive resolution. However, the overhead due to this additional calculation is usually insignificant. The reason is twofold: first, correlation computation is significantly faster at coarse resolutions, and second, we cache transformations and the sub-block samples from the previous resolution and fetch from disk only the missing samples on refinement. Since the data samples are stored in Morton order, this operation is very fast.

3.7 Multithreaded Implementation

To hide the disk I/O latency introduced by the sub-block refinement process, we overlap I/O with computations through multithreading, so that while some threads are waiting for disk I/O to complete, the system can schedule other threads that perform alignment of sub-blocks and other computations. All the sub-block pairs are added into an alignment queue, from which each of several worker threads removes one element, loads and aligns the block at the requested resolution, and pushes the result to a result queue. The worker that processes the last element from the alignment queue notifies the result thread. The result thread processes the alignment results, and furthermore decides whether it is reliable to start discarding the sub-blocks at the current resolution, as well as to identify reliable sub-blocks to refine and push only these sub-blocks back to the alignment queue at a higher resolution level. Our multithreaded implementation not only hides disk I/O latency but also takes advantage of the inherent parallelism of the problem, as sub-blocks can be processed in parallel.

4 EVALUATION

We evaluate whether or not the steps in the algorithm perform as expected and motivate choices in tunable parameters. First, we test whether our algorithm discards the correct sub-blocks at each resolution by measuring the error between the sub-blocks transformation and the baseline best transformation using the L2 norm. We compute the sub-blocks transformation for the purpose of this test at the full resolution to accurately measure the error (although we do not use the full resolution transformation to make the decision to discard the sub-block during the coarse-to-fine alignment). For our algorithm to succeed, we expect to discard the unreliable blocks at coarse resolutions early and only refine blocks that matter the most to align the overlap at full resolution. Next, we test how our algorithm performs when only a small percentage of the sub-blocks are refined at full resolution and whether there is a significant improvement to the accuracy when a higher percentage of blocks are refined at full resolution. Finally, we test the usefulness of skipping resolution levels and using the 2D approximation for alignment at full resolution by measuring the loss in quality and the gain in terms of speed-up. We perform the evaluation by measuring performance and accuracy while aligning 77 tiles of labeled axons, arranged on a 7x11 grid with 15% overlap. The extent of each tile is 2048x2048x1000. We subdivide the overlap region into sub-blocks of size 128x128x64 (note the sub-blocks on the boundary are usually of smaller size) and align sub-block pairs in parallel. The choice of sub-block size does not significantly impact the result (we discuss this in more detail in Sect. 5.1).

Baseline Best Transformation. In the absence of an actual ground truth to compare our results against, we refer to the best transforma-

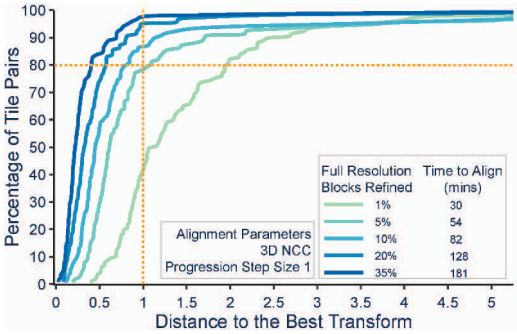

Figure 7: Alignment quality curves demonstrate that our solution is robust even when only 1% of the blocks are refined at full resolution. The vertical and horizontal lines assist in comparing the different levels of refinement with respect to quality. A significant number of tile pairs improve in quality going from 1% to 5%, and then we run into diminishing returns in the rate of improvement. Refining 1% of the overlap aligned 80% of tile pairs with at most 2 voxel L2 error with respect to the baseline best transform.

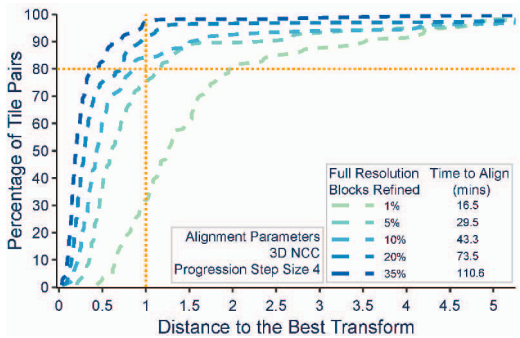

Figure 8: Alignment quality curves demonstrate using progression step size 4 has a very small impact on the quality. However, it significantly improves the speed ($1.65-1.83$x). Refining 1% of the overlap still aligns 80% of tile pairs with at most 2 voxel L2 error; however, a larger percentage of tile pairs have errors between 2.5 and 4 voxels. The larger the distance to the best transform, the higher the L2 error and hence the lower the quality.

tion — that aligns most of the overlap — from the full resolution exhaustive search as the ground truth. To compute the ground truth, we subdivide the overlap region at full resolution into sub-blocks of size $128 \times 128 \times 64$ and align each sub-block pair. We group the sub-blocks with unique transformation into clusters and order the resulting clusters (highest to lowest) using cluster size, breaking ties using the average reliability score of all the sub-blocks within a cluster. We discard clusters with an average reliability score of less than 20% as unreliable. From the remaining clusters, we choose the most reliable sub-block from the largest cluster as the candidate block for the final transformation. Next, we select additional candidates from the remaining clusters whose relative transformation distance is less than or equal to five voxels (empirically chosen) to the candidate sub-block's transformation. We define the centroid of the candidates' transformation as the *best transformation* between the pair of tiles. Additionally, we visually validate the quality of the best transform.

When to Discard Blocks. Fig. 5 illustrates at what stage our approach discards a sub-block and what is the error between the sub-block's transformation (computed at full resolution) and the baseline best transformation (hereon, we refer to this as the ground truth). Each point in Fig. 5 represents a sub-block, and the color encodes the resolution at which the sub-block was discarded. For all tile pairs, the sub-blocks that are furthest from the ground truth are discarded early at coarse resolutions, and only the sub-blocks guaranteed to give a good result are refined.

Accuracy vs. Percentage Refinement. We measure the alignment quality as the L2 norm between the centroid of the candidate blocks transformation to the baseline best transform. Computing the alignment with 3D NCC by utilizing all the resolution levels to identify the full resolution blocks progressively provides the most accurate result. Fig. 7 reports the alignment quality using 3D NCC and all the resolution levels. Refining 1% of the sub-blocks at full resolution aligned 42% of the tile pairs with at most 1 voxel error and 94% of the tile pairs with at most 3 voxel error. Refining 35% of the blocks at full resolution resulted in 97% of the tile pairs with at most 1 voxel error. Although the quality significantly improved, the runtime was roughly 6 times slower than refining 1% of the blocks. However, for most datasets, refining 1-5% of the blocks at full resolution is sufficient. As resolution increases, computing NCC in 3D becomes expensive. We evaluate next the use of progression step size and its impact on the runtime and quality.

Skipping Resolution Levels. We use progression step size to indicate the number of resolution levels skipped during refinement. With progression step size equal to 4, blocks are coarsened/refined at every 4th resolution. Although progression step size greater than 1 offers a guaranteed speed-up, the percentage of sub-blocks refined at any particular resolution still depends only on the prune-start resolution, total number of sub-blocks, and the number of sub-blocks to align at full resolution. Therefore, at each refinement, we discard a higher percentage of sub-blocks; consequently, the quality of the sub-blocks that are refined at higher resolutions may change. Using progression step size too aggressively can result in reliable sub-blocks being discarded even before their features are resolved and instead are replaced by false positives. To ensure reliable quality, we skip resolution levels only after the *prune-start resolution*. Fig. 8 reports the alignment quality with progression step size equal to 4. Skipping resolutions $l-3$ and above results in a speed-up of $1.65-1.83$, with almost identical alignment quality (as progression step size equal to 1). We recommend using progression step size equal to 4 by default and switching to a lower step size only when noise levels are very high.

Speed vs. Quality. Fig. 9 reports the alignment quality with progression step size equal to 4, using MIP 2D NCC at full resolution. Using the 2D approximation resulted in a speed-up of $6-16$x, compared to using 3D NCC with progression step size equal to 1. However, refining 1% of the blocks at full resolution resulted in only 20% of the tile pairs being aligned with at most 1 voxel error (in comparison to 42% with 3D NCC). In contrast, refining 20% of the blocks aligned 80% of the tile pairs with at most 1 voxel error (in comparison to 94% with 3D NCC). We refer to switching to 2D MIP NCC at full resolution as hybrid mode. When we switch to the 2D approximation at a lower resolution, we specify the resolution level at which the algorithms are switched. In the hybrid mode, refining 20% of the blocks is only two times more expensive than refining 1% of the blocks. Incidentally, refining 5% of blocks with progression step size equal to four (second curve from the bottom in Fig. 8), using only 3D NCC, runs at a comparable runtime vis-à-vis refining 1% of blocks using 3D NCC and all the resolutions (first curve from the bottom in Fig. 7), although with significantly higher quality. Refining 20% of blocks with progression step size equal to four and using the hybrid mode (topmost curve in Fig. 9)) runs three times faster and has better quality. Given a fixed time budget (under 10 minutes), using the hybrid mode, progression step size equal to four, and refining at least 20% of the blocks at full resolution produce the most accurate solution. Given a fixed error bound (1 micron), the fastest time to solution is roughly 43 minutes.

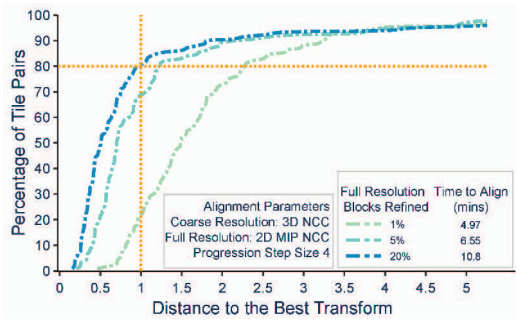

Figure 9: Alignment quality curves shows using the hybrid mode the quality reduces by at most 0.5 voxel. Refining 1% of the overlap aligns 80% of the tile pairs with at most voxel 2.5 voxel L2 error. The speed-up however is significant. Refining 1% of overlap is 3.3x faster in comparison to 3D NCC with progression step size 4, and is 6x faster vs. 3D NCC with progression step size 1.

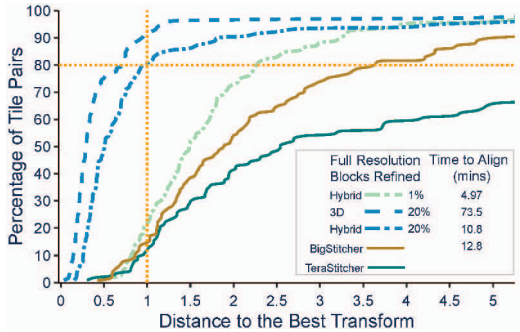

Figure 10: Alignment quality curves show how our various modes compare with the state of the art. Compared to BigStitcher, our fastest mode (hybrid 1%) is 2.57x faster and produces a higher quality result. Our more accurate mode (hybrid 20%) runs at a similar time to the state of the art, but with a significantly lower error. Our most accurate mode (3D 20%) runs 5.75x slower. Our memory footprint is 6.5x smaller than that of BigStitcher.

5 Results

For performance and accuracy evaluation, we used two hardware configurations: 1) a desktop with Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz, 3501 Mhz, 6 Core(s), and 2) a server with 144 core Intel(R) Xeon(R) CPU E7-8890 v3 @ 2.50GHz with 1TB RAM. First, we analyze the impacts of tunable parameters, such as sub-block size and consistency threshold. These results yield configurations trading quality for accuracy, and we compare their performance against state-of-the-art approaches, namely BigStitcher [19] and TeraStitcher [5].

5.1 Effects of Tunable Parameters

We evaluate the effects of tunable parameters by aligning the 77 tile dataset using a single configuration — 20% of the sub-blocks refined at the full resolution, progression step size equal to 4, and using hybrid mode.

Choosing a Good Sub-Block size. We discuss the relationship between sub-block size, alignment quality, and runtime using the results reported in Table 1 and provide guidelines on how to tune this parameter. We refer to the different sub-block sizes in Table 1 using the labels S_0, S_1, S_3, and S_4. We chose the sub-blocks sizes based on the maximum expected translation in our test data, which was 40 voxels in the x-direction, 80 voxels in the y-direction, and 20 voxels in the z-direction. Choosing a small block size typically results in fast data access and compute time. However, the number of coarse resolution samples that overlap the block is considerably small. As a result, features are often underdeveloped at coarse resolutions, leading to random correlations. Therefore, a high level of refinement is usually required to find a stable resolution with sufficient informative sub-blocks to reliably discard them. In contrast, a large sub-block size often misrepresents the alignment (false positives) at coarse resolutions due to high variations within the sub-block, which causes pruning to prematurely start early, thereby leading to large alignment errors.

We compare S_2, the optimal sub-block size for our test data, against the performance of the other choices. S_0 was 2.34x slower, with slightly lower alignment quality and for most tile pairs, the pruning of the sub-blocks only started at resolutions 5 and above. Although per sub-block compute and data fetching time was faster, the overhead due to additional searches at coarse resolutions resulted in the slowdown. In contrast, S_3 prematurely started pruning most of its blocks at resolution 1. Even though the overhead due to additional searches was minimal, the time to compute each correlation and the time to fetch the data were both more expensive. As a result, it was 1.62x slower. A nonpower of 2 sub-block size (S_1) resulted in similar alignment quality and prune-start resolution; however, it was slower by a factor of 1.24x. Since we compute the correlations in the transform domain, power of two sub-block sizes is generally faster. A good rule of thumb to use when choosing a sub-block size is it should be roughly at least twice the expected translation in each direction, with some buffer, and rounding up the result to the nearest power of two. Avoiding a very small and a very large sub-block size is generally recommended.

Choosing the Consistency-Threshold. Another tunable parameter in our algorithm is the self-consistency threshold t. We use a default value of 10% in all our experiments and did not notice a significant change in quality or runtime when t was reduced to 5%. However, a large value of t typically results in pruning starting early, leading to high false positives, and hence poor quality alignment. When we increased t to 15 and 20%, although the runtime improved by a small factor, many tile pairs started to prune sub-blocks early, resulting in an error of up to 2.5 voxels. Hence, we recommended using the default value or a slightly smaller value for t.

5.2 Performance Comparison

We compare and study the timing, accuracy, and, reliability of the first step in the alignment process - namely, phase correlation (PC) or normalized cross correlation (NCC) vs. state-of-the-art stitching tools — BigSticther and TeraSticher. We first compare the performance of our approach, BigStitcher, and Terastitcher on the desktop environment. Next, we ran our algorithm and BigStitcher, the two fastest approaches, on the server using 72 parallel threads. The timing information reported in our results includes only the time to align the data and not the time to compute any preprocessing steps, such as data format conversions.

BigStitcher configuration: BigStitcher's performance heavily relies on re-saving the data in HDF5 format (with multiple mip-map levels and user-defined chunk sizes) before starting the alignment computation. To fit the entire overlap data on the desktop's memory, we ran BigStitcher at the recommended 16x downsampled resolution and computed PC with subpixel accuracy, by checking 5 peaks and with 12 parallel tasks. On the server, we evaluate its performance at full resolution and with 16x downsampling with the same configuration as the desktop except using 72 parallel tasks. It took 3 hours and 43 minutes to convert the 77 tile data into HDF5 format with two mip-map levels (16x downsampling and full resolution).

TeraStitcher configuration: For TeraStitcher, we set the number of slices per layer to 100, the search region to 128×128, and used default values for other parameters.

Fig. 10 compares the alignment quality of our approach run with

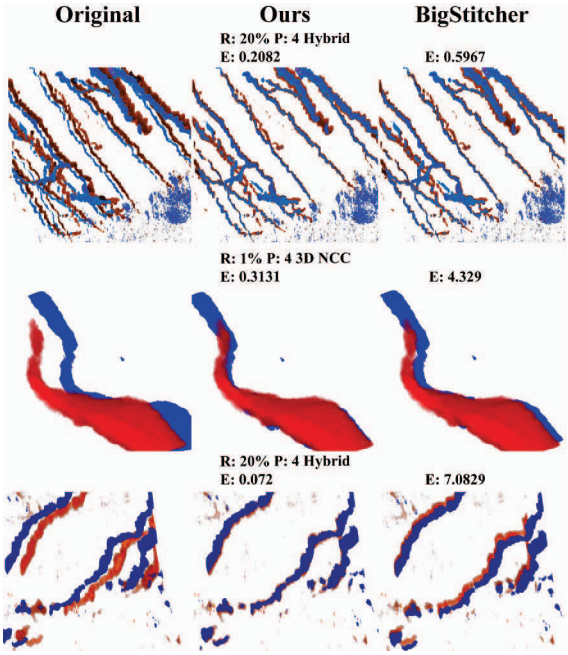

Figure 11: A visual comparison of the alignment quality using different parameters of our approach vs. the state-of-the-art BigStitcher [19]. We show the overlapping region between two tiles, rendered using contrasting color palettes (blue is tile 1 and red is tile 2) to highlight the alignment quality. We show a different tile pair on each row and report the error (E) as the L2 norm between the final transformation and the baseline best transform. For our method, the parameters are R (the percentage of sub-blocks refined at full resolution), P (the progression step size), and whether the alignment is done in hybrid or full 3D mode. Our approach in general produces better alignment results.

various configurations vis-à-vis BigStitcher and TeraStitcher. On the desktop, our fastest solution (hybrid 1%) was 1.72x faster than BigStitcher and produced a better result. Our most accurate solution with hybrid mode (hybrid 20%) was 2x slower than BigStitcher, but was more accurate. Our peak memory usage on the desktop running 12 parallel threads was 5.6GB and BigStitcher running at 16x Downsampled was 37GB. In contrast, TeraStitcher was both the slowest and the least accurate. We discuss detailed comparison numbers with BigStitcher, as it was both faster and more accurate than TeraStitcher. On the server, our fastest solution (hybrid 1%) was 2.6x faster than BigStitcher (16x downsampling) and it produced a better result. Our most accurate solution with hybrid mode (hybrid 20%) not only resulted in significantly higher quality alignment (i.e., 80% of the tile pairs aligned with at most 1 voxel L2 error, in comparison to 15.8% using BigStitcher), but also ran faster than BigStitcher (16x downsampling). Our most accurate solution (3D 20%) aligned 90% of the tile pairs with at most 1 voxel L2 error and was 2.55x faster compared to BigStitcher (full resolution). Fig. 11 provides a visual comparison of our approach run with different configurations and BigStitcher. Our peak memory usage on the server running 72 parallel threads was 7.8GB (refining 1% of the blocks) and 19.9GB (refining 20% of the blocks). In comparison, BigStitcher running at 16x Downsampled ran with 51.9GB. Additionally, our approach allows users the flexibility to use the various algorithmic modifications to trade speed for quality and vice versa. By tuning the different parameters in our algorithm, we can sweep multiple quality curves between the fastest and the most accurate setting. When working with large datasets, a small portion of the data can be aligned to identify the best parameters and the rest of the data can be aligned using the optimal parameters.

	Tunable Parameter		Effect on Performance		
	Sub-block Size (x × y × z)	n_{FS}	p_{median}	Quality (%)	Runtime (mins)
S_0	$64 \times 128 \times 32$	512	5	66.66	25.2
S_1	$100 \times 100 \times 80$	218	3	79	13.4
S_2	$128 \times 128 \times 64$	153	3	79	10.78
S_3	$256 \times 256 \times 128$	25	1	55.5	17.46

Table 1: Comparison of different sub-block sizes shows how the median prune-start resolution shifts with the sub-block size and its impact on the runtime. We report quality as the percentage of tiles pairs with Error $\leq$ 1 voxel. S_2 was the optimal sub-block size, with S_1, the nonpower of 2 sub-block size a very close second. The smallest sub-block size (S_0) was the slowest with a slight drop in quality. The biggest sub-block size (S_3) resulted in the largest error, and the poorest quality. Note: The table reports the median prune-start resolution (p_{median}) from all the tile pairs.

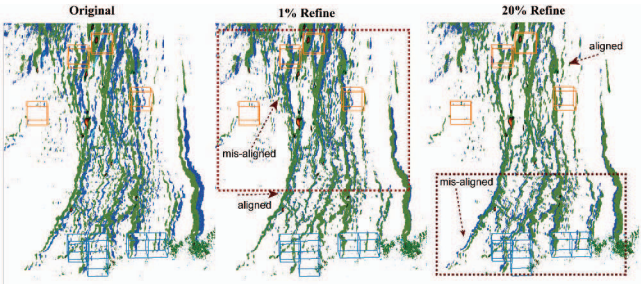

Figure 12: Small rotations can lead to multiple regions of the overlap to be partially aligned by different translations. Although it is impossible to correct for such alignment errors using a single 3D translation, changing the percentage of blocks to refine affects whether the bottom (middle) or top (right) of the stack is aligned.

6 Conclusions and Future Work

We introduced a fast iterative approach to compute coarse-to-fine alignment, identifying and refining only the sub-blocks that are likely to give good translations to align overlapping image tiles, and demonstrated both performance and accuracy improvements over prior approaches using significantly less memory. However, one of the key limitations of the current implementation is solving only for translations. In general, it is sufficient to recover 3D translations to align large microscopy volumes. However, in imaging real tissue, several problems may occur. For instance, small rotations of the sample holder, drift of the microscope stage, or even thermal expansion of the tissue can often result in more complex transformations. We encounter tile pairs in our dataset, in which such effects appear. Fig. 12 illustrates a tile pair where no single translation aligns both the top and bottom of the image stack. In this case, our approach finds more than one cluster during the two-stage stratification; however, the translation we finally chose is sensitive to the percentage of overlap refined at full resolution. Although the top cluster aligns a higher percentage of the overlap, as can been seen in Fig. 12, refining 1% of the sub-blocks at full resolution resulted in the bottom cluster being chosen, whereas refining 20% of the sub-blocks resulted in the top cluster being chosen. In future work, we will investigate affine and nonlinear transformations, replacing Euclidean distances with the matrix distance metric to use our rapid sub-block elimination method based on two-stage clustering to drive generic transformations. Other areas for future investigation are adapting the strategies to electron-microscopy and digital pathology images. In both areas, a much higher feature density might support an even more aggressive down-selection of sub-blocks.

REFERENCES

[1] D. Barnea, H. Silverman, et al. A class of algorithm for digital image registration. *IEEE Tran. on Computers*, pp. 179–186, 1972.

[2] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pp. 404–417. Springer, 2006.

[3] D. D. Bock, W.-C. A. Lee, A. M. Kerlin, M. L. Andermann, G. Hood, A. W. Wetzel, S. Yurgenson, E. R. Soucy, H. S. Kim, and R. C. Reid. Network anatomy and in vivo physiology of visual cortical neurons. *Nature*, 471(7337):177–182, 2011.

[4] A. Bria, M. Bernaschi, M. Guarrasi, and G. Iannello. Exploiting multilevel parallelism for stitching very large microscopy images. *Frontiers in Neuroinformatics*, 13, 2019. doi: 10.3389/fninf.2019.00041

[5] A. Bria and G. Iannello. Terastitcher - a tool for fast automatic 3d-stitching of teravoxel-sized microscopy images. *BMC Bioinformatics*, 13(1):316, Nov 2012. doi: 10.1186/1471-2105-13-316

[6] K. Briechle and U. D. Hanebeck. Template matching using fast normalized cross correlation. In *Optical Pattern Recognition XII*, vol. 4387, pp. 95–102. SPIE, 2001.

[7] K. L. Briggman, M. Helmstaedter, and W. Denk. Wiring specificity in the direction-selectivity circuit of the retina. *Nature*, 471(7337):183–188, 2011.

[8] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003. Nonrigid Image Registration. doi: 10.1016/S1077-3142(03)00009-2

[9] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pp. 207–212, 1984.

[10] M. EMMENLAUER, O. RONNEBERGER, A. PONTI, P. SCHWARB, A. GRIFFA, A. FILIPPI, R. NITSCHKE, W. DRIEVER, and H. BURKHARDT. Xuvtools: free, fast and reliable stitching of large 3d datasets. *Journal of Microscopy*, 233(1):42–60, 2009. doi: 10.1111/j.1365-2818.2008.03094.x

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231. AAAI Press, 1996.

[12] H. Foroosh, J. B. Zerubia, and M. Berthod. Extension of phase correlation to subpixel registration. *IEEE transactions on image processing*, 11(3):188–200, 2002.

[13] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[14] M. Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Transactions on Image Processing*, 10(4):526–533, 2001.

[15] A. Goshtasby. Template matching in rotated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(3):338–344, 1985. doi: 10.1109/TPAMI.1985.4767663

[16] A. Goshtasby, S. H. Gage, and J. F. Bartholic. A two-stage cross correlation approach to template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(3):374–378, 1984. doi: 10.1109/TPAMI.1984.4767532

[17] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE transactions on pattern analysis and machine intelligence*, 27(9):1430–1445, 2005.

[18] D. Hoang. hana - a lightweight reader and writer for idx. `https://github.com/hoang-dt/hana`, 2020.

[19] D. Hörl, F. Rojas Rusak, F. Preusser, P. Tillberg, N. Randel, R. K. Chhetri, A. Cardona, P. J. Keller, H. Harz, H. Leonhardt, M. Treier, and S. Preibisch. Bigstitcher: reconstructing high-resolution image datasets of cleared and expanded samples. *Nature Methods*, 16(9):870–874, Sep 2019. doi: 10.1038/s41592-019-0501-0

[20] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, pp. II–II. IEEE, 2004.

[21] S. Korman, D. Reichman, G. Tsur, and S. Avidan. Fast-match: Fast affine template matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2331–2338, 2013.

[22] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets. In *IEEE International Conference on Cluster Computing*, pp. 103–111, 2011.

[23] J. P. Lewis. Fast template matching. In *Vision Interface*, pp. 120–123, 1995.

[24] W. Li and E. Salari. Successive elimination algorithm for motion estimation. *IEEE transactions on image processing*, 4(1):105–107, 1995.

[25] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[26] B. Nasihatkon and F. Kahl. Multiresolution search of the rigid motion space for intensity-based registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(1):179–191, 2017.

[27] W. Ouyang, F. Tombari, S. Mattoccia, L. Di Stefano, and W.-K. Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE transactions on pattern analysis and machine intelligence*, 34(1):127–143, 2011.

[28] W. K. Pratt. Correlation techniques of image registration. *IEEE transactions on Aerospace and Electronic Systems*, (3):353–358, 1974.

[29] A. Rangarajan, H. Chui, and J. S. Duncan. Rigid point feature registration using mutual information. *Medical Image Analysis*, 3(4):425–440, 1999. doi: 10.1016/S1361-8415(99)80034-6

[30] S. Saalfeld, A. Cardona, V. Hartenstein, and P. Tomančák. As-rigid-as-possible mosaicking and serial section registration of large sstem datasets. *Bioinformatics*, 26(12):i57–i63, 2010.

[31] G. Sharp, S. Lee, and D. Wehe. Icp registration using invariant features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):90–102, 2002. doi: 10.1109/34.982886

[32] C. Studholme, D. J. Hawkes, and D. L. Hill. Normalized entropy measure for multimodality image alignment. In K. M. Hanson, ed., *Medical Imaging 1998: Image Processing*, vol. 3338, pp. 132 – 143. International Society for Optics and Photonics, SPIE, 1998. doi: 10.1117/12.310835

[33] P. Thevenaz, U. E. Ruttimann, and M. Unser. A pyramid approach to subpixel registration based on intensity. *IEEE transactions on image processing*, 7(1):27–41, 1998.

[34] F. Tombari, S. Mattoccia, and L. Di Stefano. Full-search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE transactions on pattern analysis and machine intelligence*, 31(1):129–141, 2008.

[35] P. van Oosterom and T. Vijlbrief. The spatial location code. In *International Symposium on Spatial Data Handling*, pp. 12–16, 1996.

[36] G. J. Vanderbrug and A. Rosenfeld. Two-stage template matching. *IEEE transactions on computers*, 26(04):384–393, 1977.

[37] A. Ventura, A. Rampini, and R. Schettini. Image registration by recognition of corresponding structures. *IEEE Transactions on Geoscience and Remote Sensing*, 28(3):305–314, 1990. doi: 10.1109/36.54357

[38] Y. Yu and H. Peng. Automated high speed stitching of large 3d microscopic images. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 238–241, 2011. doi: 10.1109/ISBI.2011.5872396

[39] B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003. doi: 10.1016/S0262-8856(03)00137-9