Real-time porosity mapping and visualization for synchrotron tomography

Aniket Tekawade, Viktor Nikitin, Yashas Satapathy, Zhengchun Liu, Xuan Zhang, Peter Kenesei, Francesco De Carlo, Rajkumar Kettimuthu, Ian Foster

***Abstract*—Applications of X-ray computed tomography (CT) for porosity characterization of engineering materials often involve an extended data analysis workflow that includes CT reconstruction of raw projection data, binarization, labeling and mesh extraction. It is often desirable to map the porosity in larger samples but the computational challenge of reducing gigabytes of raw data to porosity information poses a critical bottleneck. In this work, we describe algorithms and implementation of an end-to-end porosity mapping code that processes raw projection data from a synchrotron CT instrument into a porosity map and visualization in the form of triangular face mesh. Towards this objective, we report the development of a novel subset reconstruction scheme for X-ray CT using filtered back-projection and a convolutional neural network that allows us to reconstruct arbitrarily-shaped subsets of a tomography object. We build upon this scheme to implement the complete code for porosity mapping. The code first detects possible voids by performing a coarse reconstruction on down-sampled projections and then improves the shape of those voids with higher detail offered by reconstructing selected subsets from the original raw data. We report measurements of the time taken by this code to perform complete processing from raw data to a triangular face mesh for several visualization scenarios on a single high-performance workstation equipped with GPU. We show that we can now visualize local porosity within a 8 gigavoxel CT volume (12 gigabytes raw data) within 1 to 2 minutes and a 64 gigavoxel CT volume (100 gigabytes of raw data) within 3 to 7 minutes.**

***Index Terms*—X-ray Tomography, Computed Tomography, 3D convolutional neural networks, real-time analysis.**

I. INTRODUCTION

COMPUTED tomography (CT) systems typically acquire a set of X-ray projection images (or radiographs) as a sample is rotated about a known axis. These projections, plus the corresponding angles θ, can then be used to reconstruct a voxel image $f(x, y, z)$—a tomographic object—where the intensity f at each voxel coordinate x, y, z captures a measure of its opacity (more precisely, the X-ray mass absorption coefficient of the material at that coordinate). Many applications of CT involve the mapping and characterization of porosity in solid materials. Here, the voxel image shows voids within a material such as steel [1], copper [2], alloy [3], battery material [4], or nuclear fuel [5]. Based on their intensity, the void regions simply represent air or the absence of the corresponding material. But based on an analysis of the morphology and the relative location of individual voids, those void regions could be characterized as long cracks, individual round pores, or interconnected porous regions. This characterization requires additional analysis steps. First, the voxel image needs to be binarized to yield a binary intensity value, representing just either material or void, at each voxel coordinate. This binarization results in an implicit representation of the void and metal regions. An additional connected components labeling step further assigns unique labels $l(x, y, z)$ for voxels inside of voids disconnected from each other. A collection of disconnected voids and their corresponding locations in the object form an explicit map of porosity in the material which allows size and shape measurements. For visualizing this map, a subsequent marching cubes [6] step may be used to extract a triangular face mesh for rendering purposes. The mesh represents a list of faces (list of vertices and a vertex connectivity map representing the faces).

Constructing an image processing pipeline for porosity characterization is often a trial-and-error approach which involves many manual iterations of parameters and choice of algorithms to improve pore detection accuracy (minimum size of a detectable pore) in the presence of noise and artifacts [7]–[9]. Often these iterations involve jumping through dedicated software for reconstruction, segmentation, and visualization while check-pointing intermediate data that not only increase the data-storage burden but also slow down the workflow due to the repetitive read-write operations. We argue that it is possible to automatically execute the entire analysis from raw data to porosity maps with a turnaround time of a few minutes. With such fast computation we could minimize the cost of iteration for improving porosity workflows, eliminate the data burden, and allow *real-time* visualization of porosity in a material as soon as the scan is acquired. Achieving such real-time processing is crucial for realizing next-generation applications such as mosaic scanning of larger samples [10], [11] (i.e., scanning the sample at different positions followed by stitching procedures) and interactive visualization for *in-situ* studies (where one may need to visualize crack propagation as mechanical loads are progressively increased on a sample mounted on a CT system [12]).

In this work, we describe the algorithms and data structures that we have developed for fast porosity mapping and demonstrate a complete implementation that processes raw data into a face mesh for visualization. This implementation is now part of the open-source "Tomo2Mesh" project (see supporting information in Section VI). Most originally, the code includes a *voxel subset reconstruction scheme* (VSRS) that, when presented with coordinates to an arbitrary subset of voxels in tomographic object space, reconstructs the binarized intensity

All authors are affiliated with Argonne National Laboratory. A. Tekawade, Y. Satapathy, Z. Liu, R. Kettimuthu and I. Foster are with the Data Science and Learning Division. V. Nikitin, P. Kenesei and F. De Carlo are with the X-ray Science Division. X. Zhang is with the Nuclear Science Division.

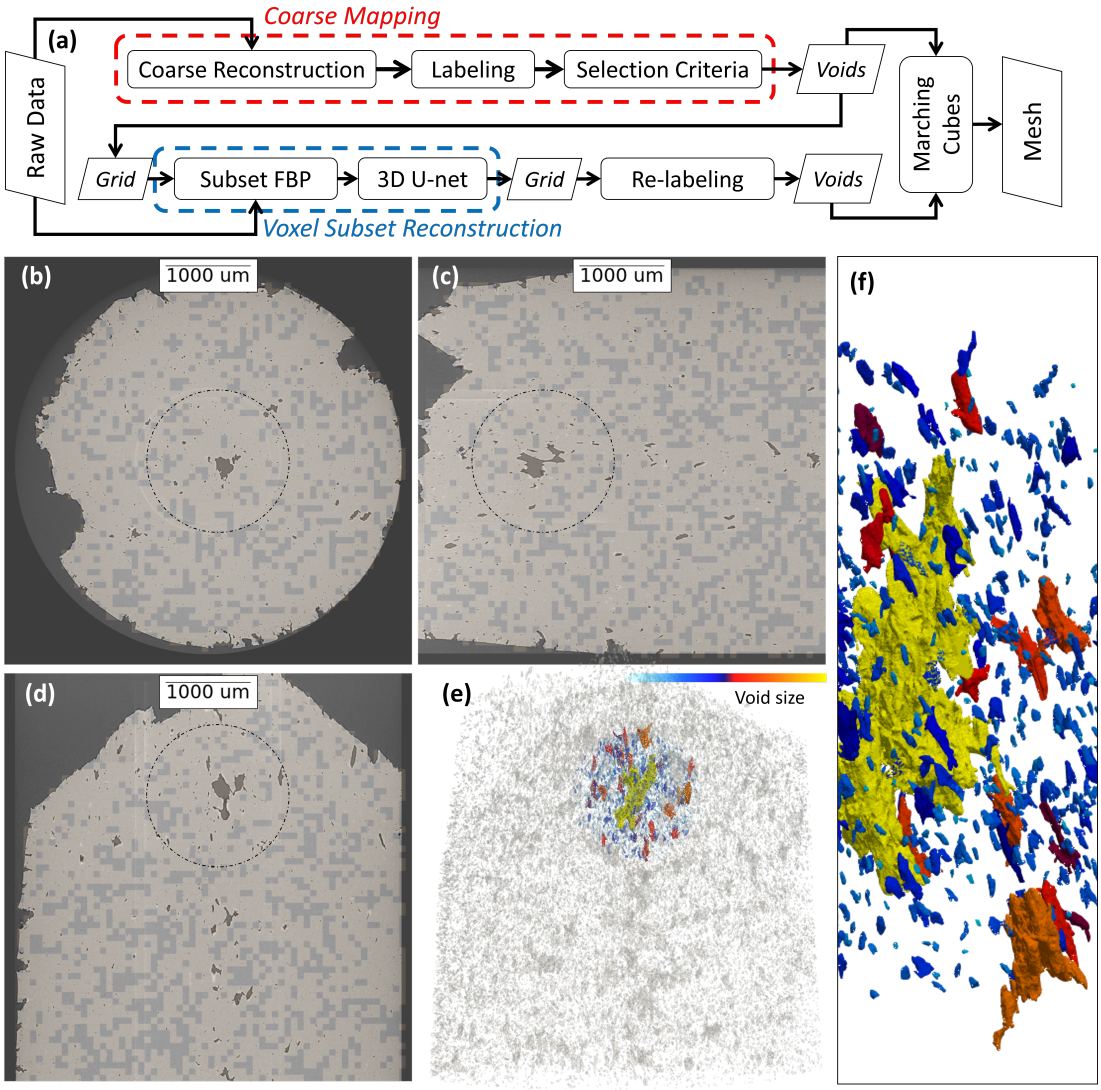

Fig. 1. Operation of the real-time porosity mapping code. (a) The various components and steps used in the code. The **Coarse Mapping** scheme generates a *coarse reconstruction* followed by a connected components step to identify and *label* disconnected voids, after which some *selection criteria* can be used to discard unwanted voids; the output is a coarse *Voids* collection. A *Grid* data structure derived from *Voids* is passed to the **Voxel Subset Reconstruction** scheme that uses filtered back projection (*Subset FBP*) and a tiny *3D U-net* to recover the selected voxel subset, outputting a list of cubic patches aligned with *Grid* to which *relabeling* is applied to generate a new detailed *Voids* collection. Finally, the *Marching Cubes* step generates a face mesh from both the coarse and detailed *Voids* collections. (b-d) Three orthogonal views drawn from the grayscale reconstruction of a CT scan of a 3D steel part with pores of various sizes. The darker regions show where the coarse mapping scheme has detected possible voids; the dotted circle delineates a spherical region of interest selected by the user around the largest pore. (e) A rendering of two overlapped triangular meshes: the coarse map of all detected voids (in gray) and the detailed map (with color indicating void size) obtained by reconstruction of the spherical-shaped subset. (f) Zoomed-in view of the detailed map of the spherical region obtained after subset reconstruction.

at those voxels, a task that it achieves by first applying filtered back-projection (FBP) to recover the grayscale intensity, and then using a compact 3D convolutional neural network (CNN) to perform denoising and binarization. To identify the voxel subset for VSRS, the regions where voids may be exist are first identified by a *coarse mapping scheme*, then VSRS confirms the existence of those voids and recovers the full voxel resolution of the detected voids. Voids are stored as a collection of 3D images with their corresponding locations in the object space. Because we only reconstruct subsets of the tomographic object space, a net computational speed-up is realized w.r.t. the baseline case where the full object is reconstructed. We show via experiment that computation time scales roughly linearly with respect to the fraction of voxels enclosed within the subset.

As compared to the best possible implementation of the baseline case (full object reconstruction, segmentation, and labeling), we show that our code is twice as fast with no loss of pore detection accuracy. Secondly, we show that we can tune the minimum detectable pore size for the coarse mapping scheme so as to detect only larger voids, with additional computational speed-up. Finally, we show that it is possible to save computation time for visualizing local porosity at full resolution by selecting arbitrarily shaped regions within the object. This feature of our code is not trivial to implement in conventional reconstruction codes.

We conduct computation time measurements on a high-performance workstation equipped with a GPU and test on two datasets (1) a *2k* volume with eight gigavoxels (12 gigabytes of raw data) and (2) *4k* volume with 64 gigavoxels (100 gigabytes of raw data). We show that our code can process raw data of a *2k* volume into a collection of voids without any loss in pore detection accuracy within four minutes and, when the minimum detectable pore size is doubled, in less than two minutes. We can, furthermore, visualize a spherical subset automatically identified around the largest void in the volume in less than one minute. While we cannot process the *4k* volume on a single workstation at the native resolution of the raw data, we show that by increasing the minimum detectable pore size by 4X, we may obtain a porosity map within six minutes. For computing a triangular mesh from the collection of voids, we provide a parallel marching cubes implementation that generates meshes in parallel for individual voids, then merges the mesh and assigns texture to it based on a morphological attribute such as total size. The mesh is made available in ".ply" format for visualization in ParaView [13] or other interactive software. For data from the *2k* volume, mesh generation time was less than 15 seconds in all cases, with mesh size less than 1 gigabyte.

In summary, we have developed an end-to-end solution for real-time porosity mapping and visualization powered by a novel voxel subset reconstruction scheme. We provide a Python-based programming interface that allows a user to configure their own visualization scenarios, for example by selecting regions where voids exist or selecting voids based on their shape.

The rest of the manuscript is organized as follows. In Section II, we discuss related work in the area of real-time tomography data-processing. In Section III, we describe the subset reconstruction scheme used to recover binarized sub-volumes within the tomography object and characterize the speed-up achieved with respect to the baseline case where the entire object is recovered. In Section IV, we apply this scheme for porosity mapping and visualization from raw scan data of a 3D printed steel part. We conclude in Section V by discussing the broader impact of this development towards bringing interactive, quantitative visualization closer to the instrument (visualization-at-the-edge) with real-time data-processing.

II. RELATED WORK

A. Real-time reconstruction

The next generation of CT applications in porosity characterization aim to tackle the long-standing trade-off between voxel resolution and field of view (FOV). For larger FOV, commercial CT systems typically compromise on voxel resolution to avoid prohibitively high scanning time and raw data size [8], [10]. For synchrotron CT systems, the scanning times are not yet prohibitive, but the prohibitively large data volumes generated when mapping larger FOV at the fundamental resolution limit has generated much interest in real-time data CT reconstruction.

Another motivation is to visualize 3D events in real-time during *in-situ* studies. Synchrotron micro-CT allows acquiring *in-situ* 3D images of pore deformation phenomena, recognized as 4D studies (fourth dimension is time). For example, Carlton et al. [14] study damage evolution mechanisms in additively manufactured steel. In fact, it was demonstrated that one could perform 3D imaging of crack propagation with one second time resolution on a synchrotron micro-CT instrument [12], thereby establishing that data-processing time, not scanning time, is the main bottleneck to achieving real-time visualization. Real-time 3D reconstruction was previously demonstrated by Bicer et al. [15] but required transferring data to a supercomputer. Buurlage et al. [16] described an *ortho-slice* reconstruction approach in which just three orthogonal slices through a predefined voxel coordinate in the object space are reconstructed. Nikitin et al. implemented a similar approach on a computer directly connected to the camera with CUDA-accelerated computing and achieved millisecond latency for reconstructing orthogonal slices [17]. However, their approach is not a substitute for full 3D reconstruction when the region of interest is not previously known.

B. Real-time segmentation and visualization

Visualizing pores, cracks, and other types of voids requires full 3D reconstruction of the object, followed by a segmentation step for labeling voids and subsequent processing into a format such as mesh object for visualization. Even when a full 3D reconstruction is achieved, the output grayscale data needs further processing to obtain a visualization of porosity. Achieving this process in real-time is complicated by the need both to automate the image processing steps and for code optimization as the data passes through multiple data structures (raw data, tomographic volume, collection of voids, polygonal mesh for visualization). Convolutional neural networks (CNN) have been employed to solve the automation problem [4], [18]–[21] and shown to be superior to conventional denoising or segmentation methods, even with limited training data. However, as we show below, conventional CNNs are a significant bottleneck in the overall workflow due to the several hundred convolution operations involved. We demonstrate that optimizing CNN architectures for short inference time and performing inference only on voxel subsets can deliver the speed-ups required for real-time processing.

III. CT RECONSTRUCTION OF VOXEL SUBSETS

Fig. 1 illustrates the porosity mapping scheme that uses subset reconstruction. Sub-figure (e) shows a rendering of two triangular meshes overlapped on each other. The gray-colored mesh is a visualization of the porosity map of all detected voids while the colored mesh shows the selected region with full detail obtained by reconstruction of the spherical-shaped subset. The novelty in our algorithm is in acknowledging that not all voxels in the volume represent useful information. Any voxel in the metal far away from the void will have the same intensity (plus noise) scaling with the attenuation coefficient of metal. If one could rapidly recover a coarse map which approximates the location and extent of the voids, then one could reconstruct only a subset of all the voxels that belong inside and around the voids, saving computation time when

compared to a complete reconstruction of the volume. In the orthogonal slices in Fig. 1, such a subset of voxels comprising of the voids is shown as a set of colored boxes ("all possible voids regions" in legend).

In the remainder of this section, we first describe the data structures we implemented to store and process such an arbitrary subset $v \subseteq V$ of the reconstructed object space V (Section III-A). Then, we describe the implementation of the following key steps: (1) filtered back-projection to recover the subset (Section III-B), and (2) binary segmentation of the subset using a convolutional neural network (Section III-C). Denoting the ratio of total number of voxels in V to those in the subset v as $r = N(v)/N(V)$, we show that the net saving in computation time scales roughly linearly with the *sparsity*, the value of $1/r$, implying that the algorithm is faster for sparser volumes. The sparsity factor is primarily influenced by the method used for selecting the subset v, referred to as the coarse mapping scheme which is described in Section IV-B.

A. Data structures for a voxel subset: Grid and Voids

Fig. 2 shows the *Grid* (P) and *Voids* (S) data structure that we use to represent a neighborhood containing voids. Such a neighborhood is simply a collection of voxels j, in which each voxel is associated with a 3D coordinate x_j, y_j, z_j; the two data structures store these coordinates in different ways. Our code involves three main steps (1) FBP, (2) binarization with 3D U-net and (3) connected components labeling of individual voids. For the FBP step (Section III-B), it is sufficient to represent this collection as a C-contiguous array (i.e., an array stored in contiguous memory addresses, in row-major order) without the need for a special data structure.

1) Grid: We used a 3D convolutional neural network similar to 3D U-net to perform the binarization step (Section III-C). 3D U-net exploits spatial locality, which means it uses context from neighboring voxels when determining whether to map the grayscale intensity of a given voxel to 0 (material) or 1 (void). The *Grid* data structure splits the voxels from the full object space into a grid of non-overlapping patches, each of size $(32 \times 32 \times 32)$, as shown by the yellow and green squares in Fig. 2. The input to U-net is the list of yellow-colored patches (those belonging to the neighborhood where voids exist). We denote the subset of voxels belonging to *Grid* as P.

2) Voids: As shown in Fig. 2, the coordinates of a bounding box that completely encloses a void may not exactly contain whole patches from the *Grid* data structure. Furthermore, the bounding box around a void may contain parts of other voids. To isolate individual voids for measuring their morphological attributes and extracting a triangular mesh separately for each void, we implemented another data structure (referred to as *Voids*) for storing a collection of bounding box coordinates and corresponding sub-volume images of isolated voids (by removing parts of other voids with a connected components labeling step). Next, we implemented methods for transferring voxel coordinates of selected voids from *Voids* to *Grid* for performing subset reconstruction and back from *Grid* to *Voids* for mapping voids between coarse and high-resolution collections for interactive visualization. *Voids* is a Python dictionary

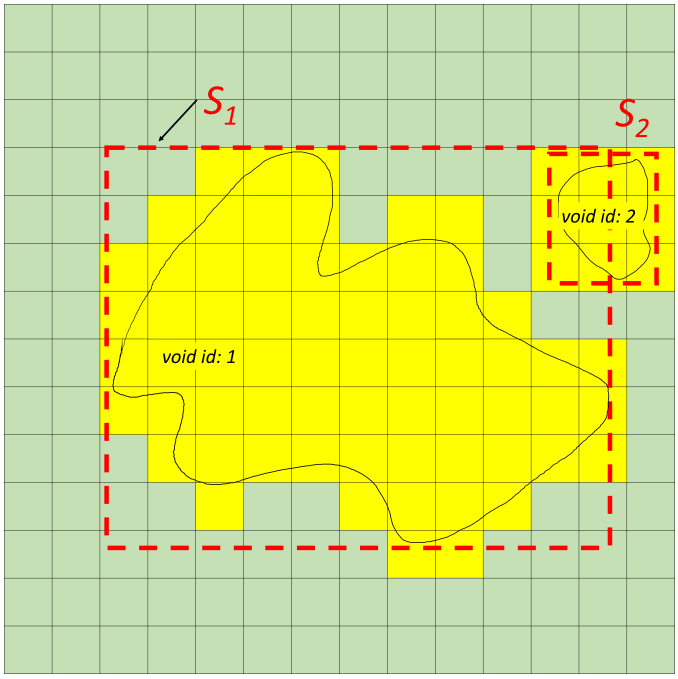

Fig. 2. 2-D illustration of the *Grid* and *Voids* data structures used to store the subset of voxels that occur within the neighborhood of detected voids. All voxels in the regular grid constitute the object space V, within which two voids have been detected, labeled with void ids 1 and 2. All $32 \times 32 \times 32$-voxel patches containing the void constitute the subset of V stored as *Grid* P (in yellow). By applying connected components labeling, the subset of the void neighborhood is first stored as a collection, *Voids*, of disconnected voids, each identified by a unique *void id* and the individual element only shows a 3D image of the void while excluding parts of other voids appearing in the bounding box (red dashed lines). Then, an instance of *Grid* data structure P is created to represent all cubic patches that could contain any part of a void.

of the following lists with each key holding the following lists: (1) *s_voids* : bounding box coordinates surrounding each void, (2) *x_voids* : the binary voxel images representing the void, and (3) *cents* : the center location (z, y, x) of the void and (4) *sizes* : net volume occupied by the void in voxels (5) specific morphological measurements for each void (e.g., *max_feret* includes calculations of Feret diameter and principal axis orientation). Each list has a length equal to the number of detected voids and each element S_{id} is indexed by a unique void id. This data structure can be easily extended to store additional information.

B. Filtered back-projection

1) Scheme: A typical reconstruction scheme starts with raw projection data, either streamed from the detector or saved to disk after acquisition. Consider a dataset of projections $d(\theta_i, s, z)$ acquired for K angles (θ_i with $i = 0, \ldots, K-1$) over 180 degrees. Each projection image is a pixel array where each row is associated with a vertical position z and each column with a signed distance s from the rotation axis. The projections are first pre-processed before applying back-projection to recover the volume. Pre-processing steps include (1) real-space filter to remove outlier pixels or detector noise, (2) correcting for flat-field deviations due to the irregular profile of the synchrotron beam, and (3) applying a convolutional

filter such as Ram-Lak, Shepp-Logan, or Parzen [22]. The filtering operation involves computing fast Fourier transform (FFT), multiplication by a convolutional filter in the frequency domain, and inverse FFT. For a convolutional filter H the procedure is as follows,

$$\tilde{d}(\theta_i, s, z) = \mathcal{F}^{-1}(\mathcal{F}(d(\theta_i, s, z)) \cdot H) \quad \forall \theta_i \in i = 0, \ldots, K-1. \tag{1}$$

Back-projection is then performed to compute a voxel value for each discrete 3D coordinate: $x_j, y_j, z_j \mapsto f_j$, where f_j is a floating point (grayscale) intensity value for index $j \in V$ of a contiguous (or flattened) array constituting the set V of all voxels within the volume defining the reconstructed object. The back-projection formula is defined as follows,

$$f_j(x_j, y_j, z_j) = \sum_{i=0}^{K-1} \tilde{d}(\theta_i, x_j \cos\theta_i + y_j \sin\theta_i, z_j) \quad \forall j \in v \tag{2}$$

This formula has previously been employed for real-time reconstruction of arbitrary orthogonal slices through the whole tomographic volume [17], [23]. Filtered back-projection was performed on a slice by fixing one of the coordinates x_j, y_j, z_j and computing Equation 2 for all indexes in the other two coordinates. While other methods that evaluate back-projection on the whole image volume may have lower computational complexity when the number of projection is large [24]–[26], the direct evaluation scheme of Equation 2 is favorable when the number of voxels for which values need to be computed is significantly smaller than the total number of voxels in the volume. Specifically, it allows for computing the intensity of individual voxels with a linear computational complexity of $\mathcal{O}(KN)$, where N is the number of voxels N and K is the number of projection images. Given a grid P of cubic patches representing the voxel subset v, we may recover the intensity $f_j(x_j, y_j, z_j)$ for all voxels in v rather than for the full volume V. The computational complexity of the back-projection step thus scales linearly with the number of voxels in the neighborhood $N(v)$, rather than the total number of voxels, $N(V)$. When this reconstruction scheme is used to recover only a subset v as opposed to the complete object V, the speed-up in computational time is high if the object is largely empty ($r \to 0$).

2) Implementation: The FBP scheme is implemented on GPU with CuPy [27], a Python library for accelerating regular algebraic operations, and with the CUDA C++ library for accelerating the back-projection operator on patches with a contiguous GPU memory access pattern. Only the sinograms required to reconstruct patches in P are transferred to GPU (CPU-GPU transfer time $t_{\text{cpu}\to\text{gpu}}$) in chunks and the filtering step is performed by using the CuPy FFT library (filter time t_{filt}). For a given P, the coordinates of voxels in it are sorted first along the global corner coordinates c_x, c_y, c_z in the reconstructed volume space, then along the local coordinates within each patch. But to ensure linear complexity for the back-projection, we first need to sort all voxel points along a contiguous array in the global coordinates of V. This operation introduces a small overhead (time for creating the mask of sorted coordinates, t_{mask}) before performing back-projection. We implemented this by first extracting a Boolean array $p_j(x_j, y_j, z_j) \leftarrow P$ from the patches in the selected neighborhood P that acts a mask to identify the index j in the contiguous array where Equation 2 needs to be evaluated (back-projection time t_{bp}). Finally, the reconstructed patches included in the voxel subset P are passed to the 3D U-net for binarization and then the binarized data is returned back to CPU memory (GPU-CPU transfer time, $t_{\text{gpu}\to\text{cpu}}$). A expression of the overall execution time for the FBP scheme (not including the U-net step) is shown in Equation 3. The binarization step using 3D U-net is described next.

$$t_{\text{fbp-gpu}} = t_{\text{mask}} + t_{\text{filt}} + t_{\text{bp}} \tag{3a}$$

$$t_{\text{fbp}} = t_{\text{cpu}\to\text{gpu}} + t_{\text{fbp-gpu}} + t_{\text{gpu}\to\text{cpu}} \tag{3b}$$

C. Tiny 3D U-net for binarization

1) Architecture: After recovering the grayscale intensity for the voxels in subset v using FBP, we semantically segment them into binary labels (void = 1, metal = 0) by passing cube-shaped grayscale patches from the *Grid* P through a 3D CNN that outputs corresponding binarized patches. We use for this purpose a CNN (see Fig. 3) based on, but significantly smaller than, the widely used 3D U-net [28].

The original 3D U-net architecture employs three analysis and synthesis blocks, with max-pooling and transpose convolution layers for down and upsampling, respectively. Each synthesis and analysis block contains two convolutional layers where the second layer has twice the number of filters. Every subsequent synthesis block also has twice as many filters in its first convolutional layer as compared to the first layer in the previous synthesis block. The first convolutional layer has 32 filters, followed by 64, 128, and so on.

Suspecting that the binarization of single-component tomography scans does not demand such a deep network, we conducted parametric studies to optimize the architecture for lowest possible inference time without loss of accuracy in the binary labeling of voids. We first determined that two analysis blocks and two synthesis blocks suffice to achieve good binarization accuracy, and result in 14% shorter inference time per voxel. Then, we determined that we could reduce the number of filters in the convolutional blocks by a factor of four, from 32 for the first convolutional layer in 3D U-net [28] to eight as shown in Fig. 3. Overall, our tiny 3D U-Net redices the time required to process a single voxel, t^v_{seg} (the total time for U-net inference divided by the number of voxels), by 7.3×, from 205 nanoseconds with original 3D U-net to just 28 nanoseconds.

2) Implementation: As this is a fully-convolutional neural network, the size of the patch passed through the network per inference step can be arbitrary [19]. From time measurements, we found that computation time per voxel is lowest for size 32 and approximately the same for larger sizes. Small input sizes allow us to define tighter neighborhoods around the region of interest. Hence the choice of 32. The binarization scheme was implemented with TensorFlow-Keras [29]. The FBP step described in Section III-B outputs a CuPy array

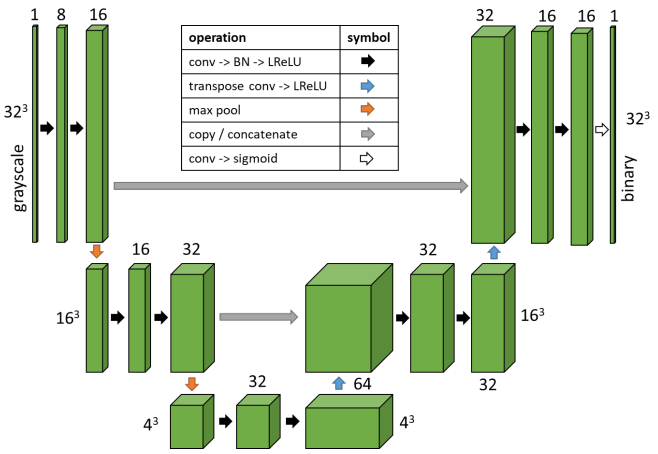

Fig. 3. Architecture diagram of the tiny 3D U-net that we used in this work, an optimized version of the much larger original 3D U-net [28].

of shape $(n, 32, 32, 32)$, where n is the number of patches. Because the Keras model prediction step is also implemented for GPU, we do a zero-copy operation and pass the data array to TensorFlow (using DLPack [30]). After binarization, the list of patches with shape $(n_{\text{patches}}, 32, 32, 32)$ is assigned to the corresponding coordinates given by P.

3) Training: For training the network, cube-shaped patches were extracted from a reconstructed volume (separate from the test dataset described here) with a corresponding manually segmented volume. These patches contain primarily empty metal regions with voids distributed in a sparse fashion. Hence, a selection criteria was applied to the extracted patches that eliminated empty regions which resulted in dramatic speed-up of convergence during model training.

4) Inference Time: As opposed to the FBP scheme, the complexity of the U-net binarization scales linearly regardless of sparsity, and can be estimated by the characteristic t^v_{seg} for a given architecture, as in Equation 4.

$$t_{\text{seg}} = r \cdot N(V) \cdot t^v_{\text{seg}} \tag{4a}$$

$$t_{\text{rec}} = t_{\text{seg}} + t_{\text{fbp}} \tag{4b}$$

D. Time Measurements for Subset Reconstruction

We performed computation time measurements on a workstation with a single NVIDIA Quadro RTX 8000 GPU (48 gigabytes of working memory) and 256 gigabytes of RAM—a compute resource that can be connected directly to a CMOS camera acquiring raw projection images [17] in a modern synchrotron tomography system. The code can also be easily deployed on multiple-GPU devices by exploiting the single-program-multiple-data (SPMD) parallelism along the z-axis that is typical for reconstruction algorithms involving synchrotron fan-beam [31].

For any step in the subset reconstruction scheme, we define speed-up as the ratio of the time taken for the baseline case ($t_{\text{subset}|r=1}$) to the time taken for a given sparsity, r. While the computational speed-up for the back-projection step itself is linear (as discussed in Section III-B), other factors limit the overall speed-up for high values of sparsity. First, the filtering step of Equation 1 must be evaluated once over the full projection width for a single sinogram at z_j even if only a single patch passes through this sinogram. Second, because the filtering step involves an FFT which is much faster on GPU, the incoming data transfer to the GPU also requires the full projection width, with cost that scales with the number of selected sinograms, regardless of sparsity. Hence, both the filtering step and the CPU-GPU data transfer become bottlenecks in the case of extremely sparse volumes.

Another factor limiting speed-up is that the effect of sparsity on speed-up is not isotropic (equal in the three dimensions). The entire FBP scheme, including filtering and data transfer, would show exactly linear speed-up *if* the same portion of each sinogram (z-slice) were recovered and only a fraction r of the total number of sinograms were chosen. However, when patches are sparse along x and y, the filtering and data-transfer steps limit the speed-up achieved in practice.

To understand this worst-case scenario, we measured computation times for different values of sparsity r, created by sampling the same number of patches, scaled by r, for each sinogram. We show in Fig. 4 the measured computation times and speed-up for both the 2k and 4k volumes, as shown in Fig. 4. Since the back-projection step is overall much slower than the filtering step, the computation time for overall reconstruction scheme does scale linearly except for very sparse volumes ($r \to 0$), where the filtering step and the CPU-GPU communication dominate. The total reconstruction time includes binarization using the 3D U-net in addition to the FBP step (Equation 4). The U-net inference time is a bottleneck for lower values of sparsity. For sparsity greater than 1/20 ($r = 0.05$), the FBP time becomes the bottleneck since the filtering step dominates over the back-projection step. The total speed-up is over 40× at $r = 0.01$ for both 2k and 4k volumes. We measured the baseline subset reconstruction time $t_{\text{subset}|r=1}$ to be 5.8 minutes for the *2k* volume and 58 minutes for the *4k* volume. The speed-up realized by our subset reconstruction scheme in real applications may be greater than these reported values because these measurements were on data with no sparsity along the z dimension, meaning that all sinograms in the data were passed from CPU to GPU and were filtered.

IV. REAL-TIME POROSITY MAPPING

Our primary claim in this work is that the *local reconstruction of voxel subsets of interest can provide the significant speed-ups required to allow real-time data processing from raw projection data into porosity visualization and measurements* (or porosity maps). By porosity map, we mean a collection of 3D images representing disconnected voids with a corresponding list of bounding box locations within the tomography object. By visualization, we mean the generation of a triangular mesh representing pores within the material. The faces in the triangular mesh can be colored based on morphological measurements of the void or their position within the object's coordinate system.

The subset reconstruction scheme outlined in Section III operates on some pre-selected voxel subset v of the full

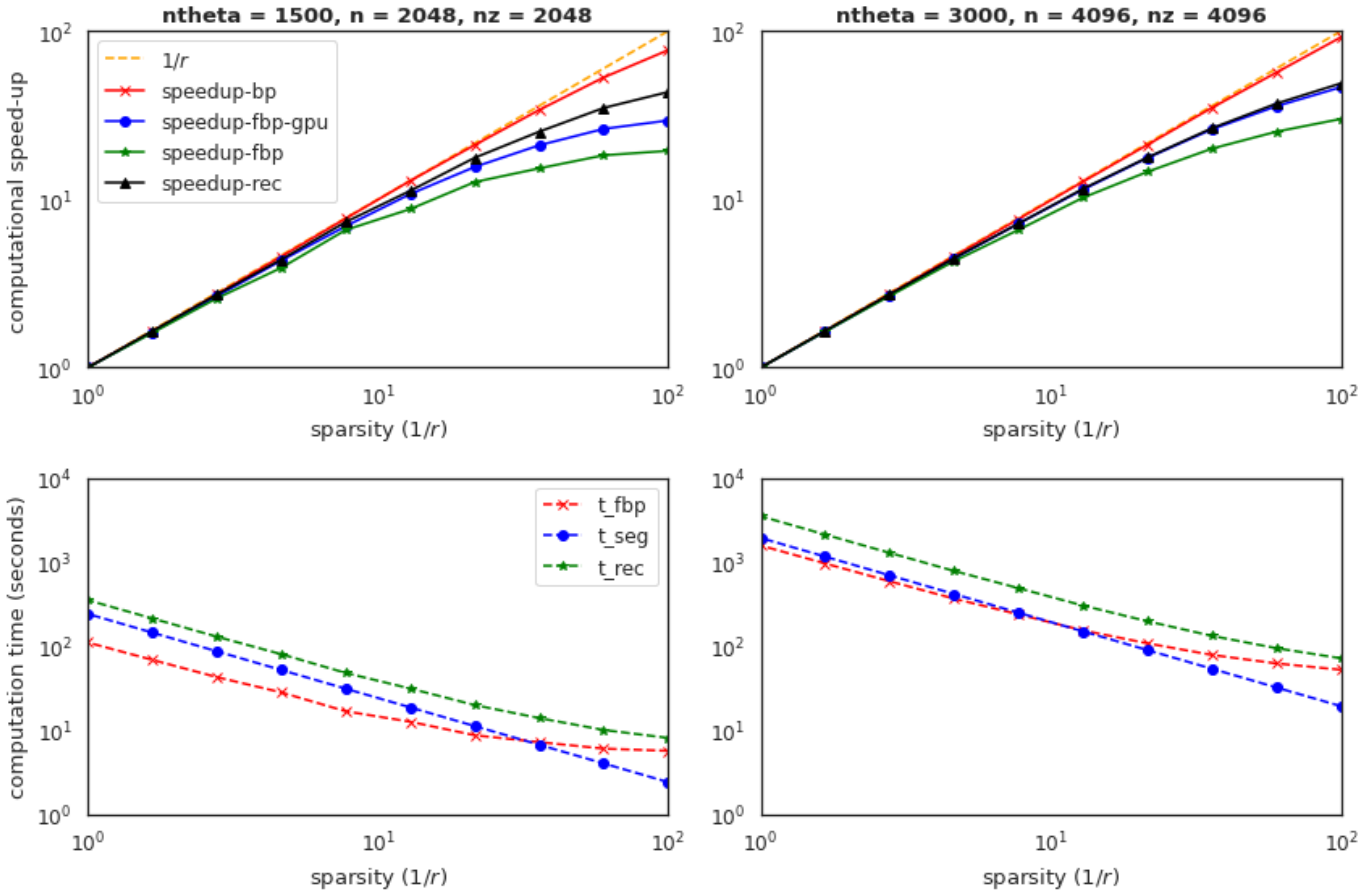

Fig. 4. Speed-ups relative to baseline time for $r = 1$ (above) and elapsed time (bottom) of the subset reconstruction scheme for different values of sparsity, $1/r$, and two volume sizes (2k: left, 4k: right). In each sub-figure, `bp` denotes back-projection; `fbp` the FBP step including CPU-GPU communication; `fbp-gpu` the FBP step, not including CPU-GPU communication; and `rec` the total time (both FBP and binarization with the tiny 3D U-net).

tomography object V and outputs binarized patches aligned with the coordinates of *Grid* P. In a typical sample, such as that depicted in images (b)-(d) of Fig. 1, much of the reconstructed volume contains metal with voids of varying size distributed sparsely throughout the volume. We may expect that, if the fraction of voxels belonging to the porous neighborhood is $r = N(v)/N(V)$, the overall computation time should be reduced in proportion to r. We may also expect that the computation time can be reduced further if we select a smaller subset that represents, for instance, the neighborhood around the largest void. However, the subset reconstruction scheme cannot proceed without first detecting these subsets and computing the coordinates of *Grid* P. Furthermore, the output of the subset reconstruction scheme is a list of sub-volumes aligned with P that do not actually distinguish between individual voids. This distinction is necessary for generating a void map showing individual voids colored by their morphological measurements.

There are thus additional implementation details to address in order to provide a complete porosity mapping scheme. We discuss these details in the remainder of this section. Specifically, we describe how our coarse mapping scheme first identifies a *Voids* collection and converts it to the subset *Grid* P comprising of regions where voids may exist (i.e., excluding the empty metal region), and the re-labeling step that takes the output from the subset reconstruction to update the *Voids* data structure with improved detail for detected voids and to remove voids that were wrongly detected by the coarse mapping scheme. We conclude the section with an overall performance analysis of the porosity mapping code.

A. Dataset description

We demonstrate our porosity mapping scheme and perform our performance measurements on a dataset obtained by mosaic scanning a 3D-printed steel cylindrical specimen (nominal diameter 6-millimeter) under monochromatic hard X-ray at the Advanced Photon Source (APS) beamline 1-ID. This specimen was printed by using a GE Concept Laser Powder Bed Fusion printer, stress-relieved at 650 °C for one hour post printing and then creep-tested until rupture at 550 °C, under 275 MPa. The mosaic comprises three horizontal and six vertical positions, for a total of 18 scans. The field of view for each scan was approximately 2.2 mm wide and 1.4 mm tall, with 1.17 micrometer voxels. The mosaic scans were performed to see through the fractured surface of one of the ruptured pieces. More about the specimen and test is provided elsewhere [32].

With an exposure time of 0.13 seconds for 3000 projections, scanning this mosaic took approximately two hours of total

exposure plus other instrument-specific overheads in moving motors to reposition the sample between scans. This resulted in 3000 projections over 180 degrees sample rotation with raw image size 4096×4096 (100 gigabytes of raw data at 16-bit precision) that would reconstruct to a 3D volume with 4096^3 voxels: what we refer to as our *4k volume*.

To enable performance measurements at different data sizes, we also generated a *2k volume* by binning the raw 4k images (4096×4096→2048×2048) and halving the angular step size, resulting in 1500 raw images over 180 degrees rotation. Generating this 2k volume directly would have taken one hour of total exposure with resolution of 2.34 micrometer per pixel, with the same exposure time per projection.

B. Porosity mapping scheme

1) Coarse reconstruction: To find the neighborhood where voids could exist, we first perform a coarse reconstruction of the entire object by down-sampling the row and height dimensions of the raw projection array by a factor b and the number of projections K by a factor b_K. Then, all voxels for this volume V_b are reconstructed by Equations 1 and 2. To improve contrast, one may prefer average-pooling (or binning), i.e., averaging a bin of four pixels (b = 4) instead of down-sampling where we skip three pixels and use the intensity in the fourth pixel. We chose down-sampling because binning requires the additional computation of a mean value. From the discussion about time complexity of FBP (Section III-B), it is clear that the FBP time for V_b would be dramatically lower than for the full volume V. Because down-sampling will increase the noise-floor in the data, too many individual voxels may be misclassified as single-voxel voids in the connected components step. We apply a Gaussian filter on the back-projected volume to minimize the number of noisy voxels. Finally, Otsu thresholding [33] is used to binarize the volume.

2) Connected components labeling: The connected components step on the binarized volume labels voxels in the down-sampled binary volume with a unique *void id* by disconnected individual voids. From the labeled volume, we generate a list of bounding boxes that store the (1) position of individual voids, (2) measurements such as total void volume, size and location of the void center and (3) the binary sub-volume that represents each void by implementing a data-structure that we refer to as *Voids* (see Section III-A).

3) Selection criteria: We may optionally apply a selection criteria to *Voids* such as "*ignore voids below size of 14 micrometers*" or "*find all voids within 800 micrometers of the largest void*," etc., to down-select the number of voids. The Tomo2Mesh code allows such selection criteria to be implemented in a few lines of Python code.

4) Export Voids to Grid: From the bounding boxes S stored in *Voids*, we compute the coordinates for the *Grid* P, i.e., the corner locations of all cubic patches that contain any voids of interest. This is done by filling an empty 3D array with the shape of the full object with the binary images of the voids from the coarse reconstruction, then extracting coordinates P on a $32 \times 32 \times 32$ grid which contain the voids. This set of coordinates along with the complete raw projection data are used to reconstruct and segment the voxel subset as described in Section III. The sparsity r for a given selection P depends on the specific criteria applied.

5) Import updated Voids from Grid (re-labeling): After recovering binarized data from the subset reconstruction scheme, we create a new instance of *Voids* which extracts new binary volumes from the subset reconstruction for the corresponding *void id* detected by the coarse mapping scheme. Hence, this new *Voids* collection contains the same voids visualized with improved level of detail. If the coarse mapping scheme wrongly classifies a noisy voxel as a void and the subset reconstruction returns with no void at that location, that void is removed and the collection is updated.

C. Performance Considerations

We discuss performance trade-offs for a special visualization scenario where a coarse reconstruction and connected components step is performed to detect voids on a down-sampled object space (down-sampling factor b), then the surface detail of those voids is improved by the subset reconstruction scheme.

1) Time measurements for 2k volume: Table I shows some measured computation times and number of detected voids for a *coarse-to-fine* scenario on a 2k volume. These time measurements are for a 2k volume for different values of b where the original object space is of size $2048 \times 2048 \times 2048$. The coarse reconstruction time includes filtered back-projection followed by a Gaussian filter and Otsu thresholding. Labeling time includes the connected components algorithm and finding bounding boxes for the voids labeled after the connected components step. Both reconstruction and labeling could be done at once after transferring the projection data to GPU and only the patches containing voids were returned to CPU. The subset reconstruction time is the same as VSRS (see Section III) and includes FBP for selected patches followed by our tiny 3D U-net for binarization. To ensure that the voids detected by VSRS are aligned with the same voids detected during coarse mapping, a re-labeling step is required. We see that the reduction in computation time is significant, especially when one only wants to detect larger voids in the sample.

2) Smallest detectable void: While the topic of minimum detectable void size from X-ray CT measurements has been debated, it is generally agreed that a void that is three voxels wide in each dimension (for a volume of 27 voxels) can be reliably detected [8]. For the *2k* volume at its native voxel resolution of 2.34 μm per voxel, this corresponds to $\sim$7 μm.

It is not possible to detect from a coarse reconstruction all voids detectable at the native voxel resolution. Indeed, if a coarse reconstruction has a down-sampling factor of b, we expect the minimum detectable void to be a least $3b$ voxels wide, which gives us a minimum detectable pore size of $3 \times 4 \times 2.34 = 28.1$ μm for $b = 4$ and $3 \times 2 \times 2.34 = 14$ μm for $b = 2$. To avoid this trade-off in detection accuracy, we adopt an aggressive detection strategy where all *possible* voids—even those smaller than three voxels wide—are first detected by the coarse reconstruction and after the subset reconstructed step, we confirm if those voids are

TABLE I
TIME TO COMPUTE A COARSE MAP OF DETECTABLE VOIDS (2K VOLUME)

				t_{coarse}		t_{subset}		t_{mapping}
b	**size**	**sparsity** $(1/r)$	**voids**	**reconstruction**	**labeling**	**reconstruction**	**re-labeling**	**total**
	pixels	ratio	num.	time (seconds)				
1	2048	1.00	1.63E+05	-	-	361	80	440
2	1024	2.19	1.54E+05	8.9	22	172	28	231
4	512	6.80	3.47E+04	0.8	4.6	60	10	75

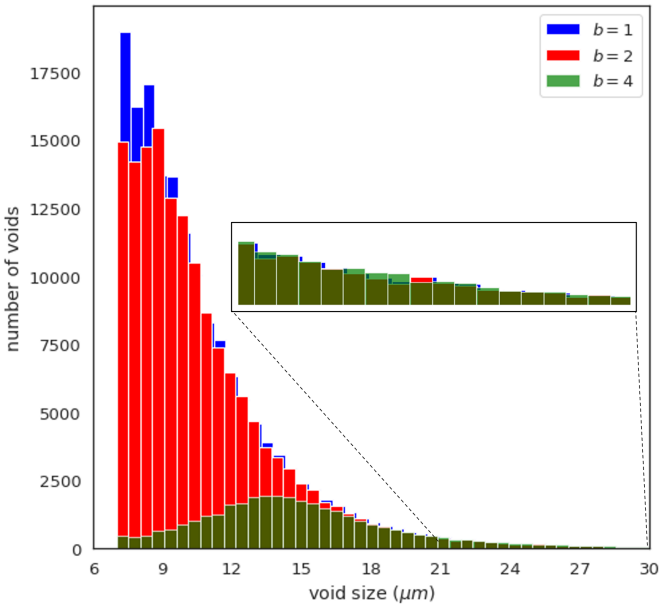

Fig. 5. Histograms of void sizes detected in the 2k volume for b=1, 2, and 4. The right tail of the histogram is clipped to show the distribution of smaller voids.

indeed real pores or artifacts. If artifacts, they are dropped from the collection. Thus, we find a workaround to the otherwise inevitable trade-off between porosity mapping time and minimum detectable void size.

We show in Fig. 5 a histogram of void size obtained for coarse maps with $b = 2$ and $b = 4$ as well as for a map at the native voxel size of 2.34 μm for the 2k volume ($b = 1$). Here, void size is defined as a cube root of the total volume enclosed in the void. We find that there is no significant loss in void detection accuracy between $b = 1$ and $b = 2$. The $b = 4$ coarse map, in contrast, misses many small voids, but does reliably detect all voids greater than 28.1 μm: predictable behavior that may be useful if a visualization of only larger voids is desired. Furthermore, we note that for a visualization of a full porosity map, smaller voids make the visualization less comprehensible; important trends in the morphology of larger voids may be revealed by ignoring them.

3) Improvements from subset reconstruction: The subset reconstruction scheme results in more accurate determination of the void surface (boundary between metal and air) because of the higher voxel resolution and the use of 3D U-net for binarization. Fig. 6 shows some comparisons between the void binarizations obtained from the coarse reconstruction scheme and the subset reconstruction. For each void detected in the coarse mapping scheme, we retain a unique void identifier (*void id*), so that the corresponding detailed reconstruction can be retrieved for any given void. Moreover, if it is determined during the subset reconstruction that a void detected from the coarse reconstruction was in fact a noisy pixel, then that void's identifier is simply removed from the *Voids* collection (see, for example, voids with identifiers 28314 and 25192 in Fig. 6). A 3D image corresponding to a void in the *Voids* collection stores binary labels within a cropped bounding box for that void while excluding other voids that may appear in the bounding box. A second connected components step after subset reconstruction ensures that each 3D image in the collection represents only the putative void isolated from its neighbors, as illustrated in Fig. 6 for void 1146.

D. Smart Visualization

1) Scenarios: A unique advantage of our subset reconstruction scheme is that it can recover arbitrarily shaped regions within the object space. To demonstrate this, we conceive of the following visualization scenarios, illustrated in Fig. 7: (1) show all voids larger than $\sim 14\mu m$ detected with the coarse-to-fine strategy described in Section IV-C, (2) locate the largest void and show a cylindrical neighborhood of 1600 μm height around its center, and (3) locate the largest void and show a spherical neighborhood of 800 μm radius around its center. To accomplish scenarios 2 and 3, all detected voids from the coarse reconstruction are further sorted by size and the *void id* of the largest void is identified. Then, a nearest neighbor search is performed on the *Voids* collection to find all voids that lie within the region (Euclidean distance for spherical neighborhood and Z-distance for cylindrical neighborhood). Table II reports measurements of time for the void mapping scheme for these scenarios for the 2k and 4k volumes. In this table, we additionally report the meshing time, t_{mesh}. We describe the implementation of this mesh generation step next.

2) Marching cubes for mesh generation: From the binarized sub-volumes of voids stored in *Voids* (from both the full reconstruction with coarse resolution and subset reconstruction at full voxel resolution), we implemented the marching cubes algorithm [6] to compute a triangular mesh object for each void which explicitly represents the surface of the void. A triangular mesh consists of (1) a list of vertices x_k, y_k, z_k that lie on the transition region between the metal and void space and (2) a connectivity map that stores a list of faces, each represented by three vertices and (3) a texture map that stores an RGB value for each vertex. From this mesh, we further

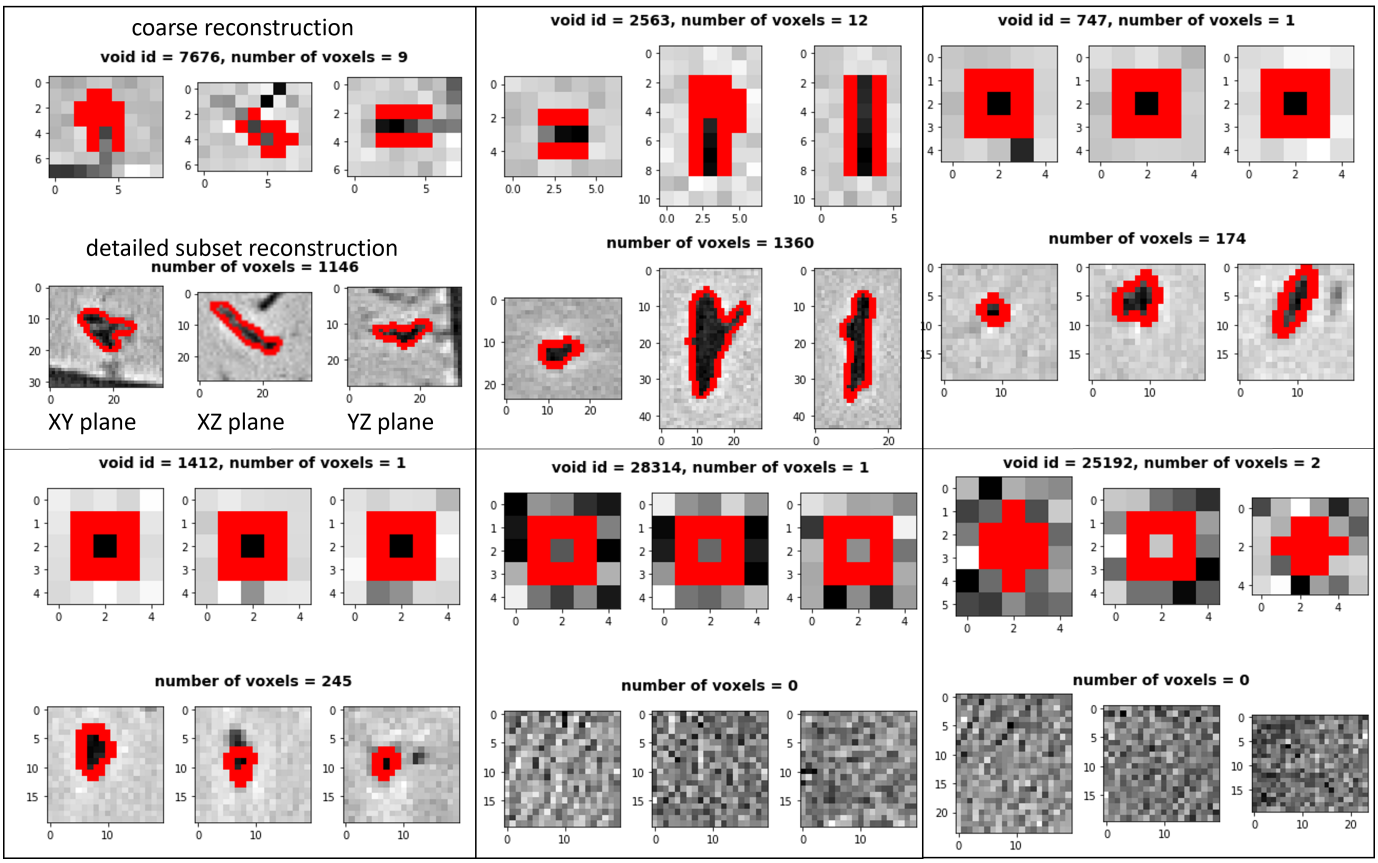

Fig. 6. For each of six voids from porosity mapping of the 2k volume at $b = 4$, we show the three orthogonal slices (left to right) for the coarse (upper) vs. detailed (lower) reconstructions. Each image shows, in red, the void's edge-map identified in the binarized output, overlaid on the corresponding grayscale intensity from the FBP output. Binarization for the coarse map is performed on a down-sampled volume (factor $b = 4$ using Otsu thresholding); for the detailed map, it is performed by using the tiny 3D U-net on subsets from the full volume. Note that in two cases (void ids 28314 and 25192), subset reconstruction reveals that a putative void in the coarse reconstruction is in fact just a noisy pixel; thus, no red pixels appear in the lower images for those two cases, and those two void identifiers are not included in the *Voids* collection. In contrast, voids with identifiers 747 and 1412 were first detected as dubious single voxels and the subset reconstruction revealed that they were definitely voids. The void with identifier 7676 appears in the neighborhood of other voids and is accurately isolated from them after the subset reconstruction.

reduce the face count by collapsing edges with length less than one pixel length. This make the mesh size manageable for better rendering. Since each void is meshed separately, we used a map-reduce scheme using Python's multiprocessing library to parallelize over the collection of voids. The complete mesh containing all voids is then exported to a ".ply" file, a format that is readable by most visualization software (we chose Paraview). For marching cubes, we included an implementation from scikit-image [34] and a subsequent step to reduce the face count by collapsing short edges from PyMesh library [35]. The reduced face count makes the mesh size manageable for better rendering.

3) Additional comments: If these scenarios were implemented with a typical fan-beam reconstruction code such as TomoPy [31], one would not see any time benefit from visualizing a cylindrical or spherical region. To reconstruct a subset of data, one simply selects the relevant range of z-coordinates to select the subset of sinograms and reconstruct them. Reconstructing a spherical neighborhood would still require reconstruction of the full cylinder, first, followed by some cropping scheme, and thus not result in any time savings. Because our subset reconstruction scheme performs reconstruction with FBP and U-net on arbitrary patches in the volume, we can realize further speed-up for the scenario involving a spherical neighborhood compared to a cylindrical neighborhood. Additionally, we note that the cylindrical or spherical regions reported in Table II are not the complete set of voxels in this region but only those voxels around the detected voids. Hence, the subset reconstruction scheme allows superior speed-up for intelligently visualizing porosity information in sparse volumes.

V. CONCLUSIONS AND FUTURE WORK

We have described a code, Tomo2Mesh, that allows for the fast visualization and measurement of porosity distribution from raw tomography data within minutes of a scan by using a high-performance workstation typically available at a synchrotron tomography beamline. As an example, a typical full CT acquisition from a micro-CT instrument such as those at beamlines 1-ID, 2-BM, or 7-BM at the Advanced Photon Source generates 1500 raw projection images from 2 megapixel CMOS cameras with shape 1920×1200 in from 1 to 30 minutes, depending on desired exposure time. Our computation time measurements on volumes of comparable

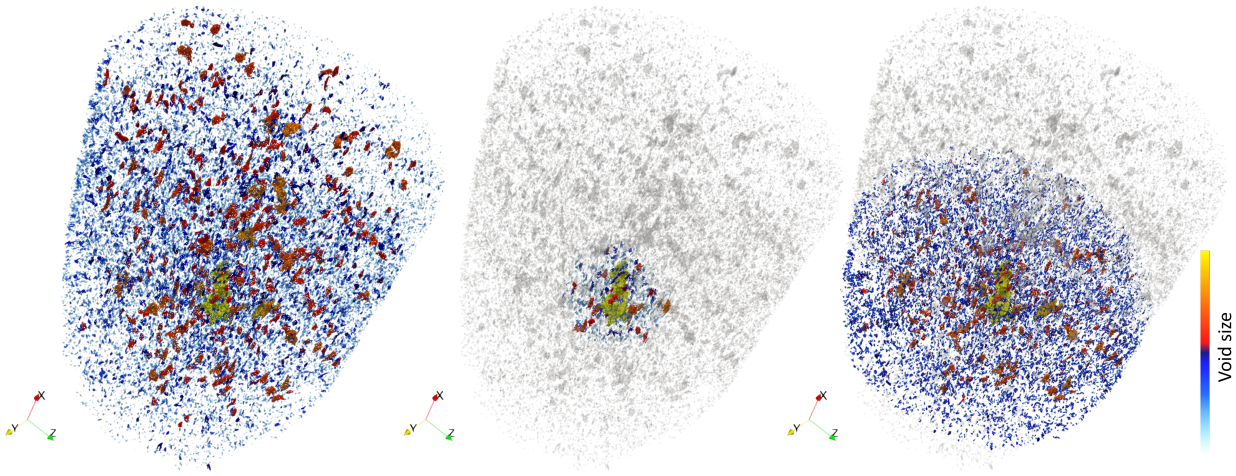

Fig. 7. Illustration of three smart visualization scenarios. In the left-most subfigure, all voids detectable from the 2k volume with $b = 2$ ($\sim 14\mu m$ or greater) are shown and colored by size. The middle and right-most subfigures show voids within cylindrical and spherical neighborhoods, respectively, centered on the largest detected void (shown in yellow).

TABLE II
TIME (SECS) TO VISUALIZE (1) ALL VOIDS LARGER THAN 14 μm, AND ALL VOIDS LARGER THAN 7 μm IN (2) CYLINDRICAL AND (3) SPHERICAL REGIONS AROUND THE LARGEST VOID

2k volume, b = 2

region and size criteria	sparsity $(1/r)$	t_{mapping}	t_{mesh}
full volume, $> 14\mu m$	7.04	103	13
cylindrical, $> 7\mu m$	6.07	106	14
spherical, $> 7\mu m$	61.5	43	8

4k volume, b = 4

region and size criteria	sparsity $(1/r)$	t_{mapping}	t_{mesh}
full volume, $> 14\mu m$	25.4	354	83
cylindrical, $> 7\mu m$	17.7	337	85
spherical, $> 7\mu m$	174	119	59

size leads us to expect that Tomo2Mesh can produce the porosity map from such an acquisition within two minutes—or less if searching for specific features such as large voids or cracks. Users can then quickly view the acquired data, correct acquisition parameters to improve the quality of measurements, or steer experiments such as for *in-situ* studies.

We hope to reduce processing time further in future work. For example, we can overlap CPU-GPU data transfer and the FBP step in the subset reconstruction scheme, consider replacing the tiny 3D U-net with a yet smaller convolutional architecture, or replace 32-bit computation with 16-bit for both FBP and CNN steps. With the availability of multiple GPU nodes, we may parallelize computation of sinograms to further reduce subset reconstruction time.

Looking forward to a time when such optimizations reduce computation times for 3D visualization from raw data to seconds, we may conceive of a novel tomography instrument that provides immersive 3D visualization on a stream of raw data without ever having to save a byte to disk. Infrastructure to stream raw projection data from tomography instruments is developing rapidly at some synchrotron beamlines [17]. Such real-time porosity mapping will allow us to automate correlative measurements that depend on micro-CT for identifying regions of interest, such as cracks or void clusters, so as to guide subsequent high-resolution diffraction, scattering, or other measurements.

VI. SUPPORTING INFORMATION

The open-source code "Tomo2Mesh" is available on GitHub (project homepage: github.com/aniketkt/Tomo2Mesh). A link to the publicly accessible Globus endpoint is provided for retrieving both the dataset and the trained U-net models used for performance measurements. To reproduce time measurements reported here, the code can be accessed via commit id 0aa7de27aa9d82c9aa611d1d09e986369f49cd27.

VII. ACKNOWLEDGMENTS

This effort was primarily funded through a Laboratory Directed Research and Development (LDRD) award titled "AI-steer: Online steering of light source experiments" from Argonne National Laboratory. Materials were provided by Oak Ridge National Laboratory under the Transformational Challenge Reactor program supported by the U.S. Department of Energy (DOE), Office of Nuclear Energy under Contract DE-AC02-06CH11357. We thank Dr. Meimei Li for support of data and guidance. Data were acquired at beamline 1-ID of Advanced Photon Source (APS). This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory, and is based on research supported by the U.S. DOE Office of Science–Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.

REFERENCES

[1] X. Zhang, "Unveiling microstructure-property correlations in nuclear materials with high-energy synchrotron x-ray techniques," *Journal of Nuclear Materials*, p. 153572, 2022.

[2] D. B. Menasche, J. Lind, S. F. Li, P. Kenesei, J. F. Bingert, U. Lienert, and R. M. Suter, "Shock induced damage in copper: A before and after, three-dimensional study," *Journal of applied physics*, vol. 119, no. 15, 2016.

[3] D. B. Menasche, P. A. Shade, J. Lind, S. F. Li, J. V. Bernier, P. Kenesei, J. C. Schuren, and R. M. Suter, "Correlation of thermally induced pores with microstructural features using high energy X-rays," *Metallurgical and Materials Transactions A*, vol. 47, no. 11, pp. 5580–5588, 11 2016.

[4] M. B. Dixit, A. Verma, W. Zaman, X. Zhong, P. Kenesei, J. S. Park, J. Almer, P. P. Mukherjee, and K. B. Hatzell, "Synchrotron imaging of pore formation in Li metal solid-state batteries aided by machine learning," *ACS Applied Energy Materials*, 2020.

[5] J. Thomas, A. Figueroa Bengoa, S. T. Nori, R. Ren, P. Kenesei, J. Almer, J. Hunter, J. Harp, and M. A. Okuniewski, "The application of synchrotron micro-computed tomography to characterize the three-dimensional microstructure in irradiated nuclear fuel," *Journal of Nuclear Materials*, vol. 537, p. 152161, 2020.

[6] W. E. Lorensen and H. E. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 163–169, aug 1987. [Online]. Available: https://doi.org/10.1145/37402.37422

[7] X. Cai, A. A. Malcolm, B. S. Wong, and Z. Fan, "Measurement and characterization of porosity in aluminium selective laser melting parts using x-ray ct," *Virtual and Physical Prototyping*, vol. 10, no. 4, pp. 195–206, 2015.

[8] A. Du Plessis, I. Yadroitsev, I. Yadroitsava, and S. G. Le Roux, "X-ray microcomputed tomography in additive manufacturing: A review of the current technology and applications," *3D Printing and Additive Manufacturing*, vol. 5, no. 3, pp. 227–247, 2018.

[9] A. Du Plessis, B. J. Olawuyi, W. P. Boshoff, and S. G. Le Roux, "Simple and fast porosity analysis of concrete using X-ray computed tomography," *Materials and structures*, vol. 49, no. 1, pp. 553–562, 2016.

[10] J. Kim, A. Slyamov, E. Lauridsen, M. Birkbak, T. Ramos, F. Marone, J. W. Andreasen, M. Stampanoni, and M. Kagias, "Macroscopic mapping of microscale fibers in freeform injection molded fiber-reinforced composites using X-ray scattering tensor tomography," *Composites Part B: Engineering*, p. 109634, 2022.

[11] M. A. Yakovlev, D. J. Vanselow, M. S. Ngu, C. R. Zaino, S. R. Katz, Y. Ding, D. Parkinson, S. Y. Wang, K. C. Ang, P. La Riviere, and K. C. Cheng, "A wide-field micro-computed tomography detector: Micron resolution at half-centimetre scale," *Journal of Synchrotron Radiation*, vol. 29, no. 2, 2022.

[12] E. Maire, C. Le Bourlot, J. Adrien, A. Mortensen, and R. Mokso, "20 Hz X-ray tomography during an in situ tensile test," *International Journal of Fracture*, vol. 200, no. 1, pp. 3–12, 2016.

[13] J. Ahrens, B. Geveci, and C. Law, "ParaView: An end-user tool for large data visualization," *The Visualization Handbook*, vol. 717, no. 8, 2005.

[14] H. D. Carlton, A. Haboub, G. F. Gallegos, D. Y. Parkinson, and A. A. MacDowell, "Damage evolution and failure mechanisms in additively manufactured stainless steel," *Materials Science and Engineering: A*, vol. 651, pp. 406–414, 2016.

[15] T. Bicer, D. Gursoy, R. Kettimuthu, I. T. Foster, B. Ren, V. De Andrede, and F. De Carlo, "Real-time data analysis and autonomous steering of synchrotron light source experiments," in *13th International Conference on e-Science*. IEEE, 2017, pp. 59–68.

[16] J.-W. Buurlage, F. Marone, D. M. Pelt, W. J. Palenstijn, M. Stampanoni, K. J. Batenburg, and C. M. Schlepütz, "Real-time reconstruction and visualisation towards dynamic feedback control during time-resolved tomography experiments at TOMCAT," *Scientific Reports*, vol. 9, no. 1, pp. 1–11, 2019.

[17] V. Nikitin, A. Tekawade, A. Duchkov, P. Shevchenko, and F. De Carlo, "Real-time streaming tomographic reconstruction with on-demand data capturing and 3D zooming to regions of interest," *Journal of Synchrotron Radiation*, vol. 29, no. 3, May 2022. [Online]. Available: https://doi.org/10.1107/S1600577522003095

[18] Z. Liu, T. Bicer, R. Kettimuthu, D. Gursoy, F. De Carlo, and I. Foster, "TomoGAN: Low-dose synchrotron x-ray tomography with generative adversarial networks: discussion," *JOSA A*, vol. 37, no. 3, pp. 422–434, 2020.

[19] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[20] Z. Su, E. Decencière, T.-T. Nguyen, K. El-Amiry, V. De Andrade, A. A. Franco, and A. Demortière, "Artificial neural network approach for multiphase segmentation of battery electrode nano-CT images," *npj Computational Materials*, vol. 8, no. 1, pp. 1–11, 2022.

[21] S. Niverty, H. Torbatissaraf, V. Nikitin, V. De Andrade, S. Niauzorau, N. Kublik, B. Azeredo, A. Tekawade, F. De Carlo, and N. Chawla, "Computational imaging in 3D x-ray microscopy: Reconstruction, image segmentation and time-evolved experiments," in *IEEE International Conference on Image Processing*. IEEE, 2021, pp. 3502–3506.

[22] F. Natterer, *The Mathematics of Computerized Tomography*. SIAM, 2001.

[23] J.-W. Buurlage, H. Kohr, W. J. Palenstijn, and K. J. Batenburg, "Real-time quasi-D tomographic reconstruction," *Measurement Science and Technology*, vol. 29, no. 6, p. 064005, 2018. [Online]. Available: https://doi.org/10.1088/1361-6501/aab754

[24] A. Dutt and V. Rokhlin, "Fast Fourier transforms for nonequispaced data," *SIAM Journal on Scientific computing*, vol. 14, no. 6, pp. 1368–1393, 1993. [Online]. Available: https://doi.org/10.1137/0914081

[25] G. Beylkin, "On applications of unequally spaced fast Fourier transform," *Mathematical Geophysics Summer School (Stanford Univ., Stanford, 1998)*, 1998. [Online]. Available: https://amath.colorado.edu/faculty/beylkin/papers/applusfft.pdf

[26] F. Andersson, M. Carlsson, and V. V. Nikitin, "Fast algorithms and efficient GPU implementations for the Radon transform and the back-projection operator represented as convolution operators," *SIAM Journal on Imaging Sciences*, vol. 9, no. 2, pp. 637–664, 2016.

[27] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "CuPy: A NumPy-compatible library for NVIDIA GPU calculations," *31st Conference on Neural Information Processing Systems*, vol. 151, 2017.

[28] O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention*. Cham: Springer, 2016, vol. 9901, pp. 424–432.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2016, pp. 265–283.

[30] "DLPack: Open in memory tensor structure," 2022. [Online]. Available: https://github.com/dmlc/dlpack

[31] D. Gürsoy, F. De Carlo, X. Xiao, and C. Jacobsen, "TomoPy: A framework for the analysis of synchrotron tomographic data." *Journal of Synchrotron Radiation*, vol. 21, no. Pt 5, pp. 1188–1193, 2014.

[32] M. Li, W.-Y. Chen, and X. Zhang, "Effect of heat treatment on creep behavior of 316L stainless steel manufactured by laser powder bed fusion," *Journal of Nuclear Materials*, vol. 559, p. 153469, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022311521006899

[33] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[34] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: Image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: https://doi.org/10.7717/peerj.453

[35] "Pymesh documentation," https://pymesh.readthedocs.io/en/latest/, accessed: 2022-07-01.