IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 July 2023, accepted 19 July 2023, date of publication 26 July 2023, date of current version 2 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3298887

RESEARCH ARTICLE

Real-Time 3D Ultrasound Reconstruction Using Octrees

CÉSAR VICTORIA[1], FABIAN TORRES[2], EDGAR GARDUÑO[3], (Member, IEEE), FERNANDO ARÁMBULA COSÍO[4], AND ALFONSO GASTELUM-STROZZI[5]

[1]Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Mexico City 04510, Mexico
[2]Laboratorio de Física Médica, Instituto de Física, Universidad Nacional Autónoma de México, Mexico City 04510, Mexico
[3]Department of Computer Science, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Mexico City 04510, Mexico
[4]Unidad Académica del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas en el Estado de Yucatán, Universidad Nacional Autónoma de México, Mérida 97357, Mexico
[5]Instituto de Ciencias Aplicadas y Tecnología, Universidad Nacional Autónoma de México, Mexico City 04510, Mexico

Corresponding author: Edgar Garduño (edgargar@ieee.org)

The work of César Victoria was supported by the Consejo Nacional de Ciencia y Tecnología (CONACyT)-Mexico Graduate Program under Doctoral Scholarship under Grant 487646. The work of Fabian Torres was support by the Postdoctoral Fellowship granted by CONACyT under Grant CVU 298645.

ABSTRACT Ultrasound is a major medical imaging modality that is widely used in healthcare because of advantages such as the use of nonionizing radiation, ease of operation, real-time imaging from different perspectives, and low operation costs. The most common ultrasound modality produces two-dimensional images (2D) with a 1D-array transducer. However, in recent years, three-dimensional ultrasound (3D US) imaging has become increasingly relevant. There are many reasons behind this shift. For example, 3D US images are easier to register with 3D images from another modality while patients undergo procedures or during presurgical planning. In particular, 3D freehand ultrasound (FUS) imaging yields 3D US images of large anatomical regions at low cost. An area of interest is scanned with a conventional 1D-array transducer, which is tracked with an attached device; the resulting 2D US images are input into a reconstruction algorithm; and the brightness values are assigned to a 3D image. Several 3D reconstruction algorithms in FUS imaging have been proposed and clinically used, and in the present work, we report a new neighbor search-based approach for reconstructing 3D FUS images based on hierarchical octrees with Morton key coding that can be implemented on GPUs using CUDA® kernels to exploit multithreading. Our approach achieves considerably faster throughput for high-resolution 3D images and can reconstruct 3D US images with dimensions of $128 \times 128 \times 128$ voxels in approximately 0.5 s. The proposed approach is a viable option for obtaining 3D US images in real time based on sets of freehand 2D ultrasound images acquired with 1D-array transducers.

INDEX TERMS 2D and 3D ultrasound, CUDA, morton key, octree, reconstruction.

I. INTRODUCTION

Ultrasound (US) has become one of the most commonly used medical imaging modalities in hospitals and clinics and is the basis for one in five medical images used for diagnoses [1]. Whereas magnetic resonance imaging (MRI) and X-ray computed tomography (CT) directly produce 3D images, conventional handheld 1D-array US transducers produce only 2D images. Furthermore, the 3D images produced by the former modalities typically cover large areas of interest in patient anatomies, with resolutions of up to 0.5 mm for CT and 1 mm for 3 T MRI machines, while 2D US images show only small sections of the human body. Nonetheless, US has various advantages over other medical imaging modalities, such as the use of nonionizing radiation, ease of operation, the ability to obtain real-time images from different perspectives, increased safety, and significantly lower acquisition and operation costs [2]. In addition to these reduced costs,

The associate editor coordinating the review of this manuscript and approving it for publication was Essam A. Rashed.

the size of US devices (equipment portability) makes them flexible and suitable for various procedures and studies in clinical environments. Hence, US has become an alternative intraoperative imaging modality that obtains information that is complementary to other modalities, such as MRI and CT.

Nevertheless, when using 1D-array US transducers, it is challenging to select an appropriate plane to visualize the desired internal structures. Furthermore, because these transducers produce only 2D images, radiologists and physicians need to integrate several images to visualize 3D anatomical information from patients. While this process is sufficient for many clinical situations, it is unacceptable when US images need to be registered with 3D images from another modality while patients undergo surgical procedures (e.g., in intraoperative guidance dual-modality coregistration or augmented reality-assisted procedures). Hence, real-time 3D US image reconstruction is necessary to avoid needing new 3D images from patients (either to refocus on a region of interest or to scan a new region). Therefore, the reconstruction speed and image quality are both relevant factors for 3D US image reconstruction in clinical environments. 3D ultrasound imaging has been used for many medical procedures, including vascular imaging, cardiology, obstetrics, neurosurgery, and image-guided surgery. Consequently, to address the limitations and drawbacks of 2D US imaging, there have been several efforts to develop 3D US imaging techniques [3], which can be grouped into three categories based on the applied hardware: 2D-array transducers, mechanical probes, and 1D-array transducers.

Mechanical probes are not widely commercially available. In addition, they are bulky and difficult to adjust because of the mechanical framework necessary for automatic scanning. Additionally, such machinery frequently introduces artifacts in the final 3D images. Moreover, 2D-array transducers are complicated to manufacture because they require numerous small elements without crosstalk, increasing the difficulty of impedance matching (very small elements lead to much larger impedance than linear phased arrays) [4]. Consequently, the volume and spatial resolution must be balanced because more piezoelectric crystals need to be included to achieve stricter impedance matching in 2D arrays [3]. Therefore, a single 2D image slice obtained by a 2D-array US transducer has much lower image quality than a 2D image slice obtained by a 1D-array US transducer. As a result, 2D-array transducers are rarely used in clinical settings [5].

As an alternative, researchers have developed 3D US imaging systems using existing conventional 1D-array transducers, which are available in almost all clinical environments, to address the drawbacks and limitations of mechanical probes and 2D-array transducers. This approach, known as 3D freehand ultrasound (FUS), has become a reliable supplementary imaging modality in the clinic because of the potential of offline reconstruction (i.e., reconstruction not performed in real-time), the potential to acquire a large field of view (FOV), the ability to investigate diverse and nonconventional imaging planes, reduced operator dependence in the acquisition process, and the wide availability of 1D-array US transducers in clinical environments. Accordingly, 3D FUS has been successfully implemented in clinical procedures such as liver biopsies, neurosurgery, region-of-interest selection in radiation therapy, and spine interventions [6], [7], [8]. Nevertheless, despite these advantages, the quality of the produced 3D images and the associated computing time are still challenges in 3D FUS [9].

In recent years, medical image processing based on graphics processing units (GPUs) has become increasingly popular because this kind of hardware is affordable and capable of computing many complex mathematical functions and methods; thus, sophisticated algorithms can be implemented and computationally difficult tasks can be executed quickly in clinical settings [10]. As a result, this technology has led to the successful implementation of real-time, or near-real-time, 3D FUS reconstruction methods [11], [12]. To take full advantage of GPUs, an appropriate and efficient reconstruction algorithm is crucial, and the development of such an algorithm is one of the most important tasks in 3D FUS using GPUs [5].

In this paper, we propose a new method for optimizing 3D FUS reconstruction based on neighbor search to quickly produce 3D US images without reducing image quality. This optimization is achieved by using hierarchical octrees in combination with Morton order indexing, which is also referred to as Z-order indexing [13], resulting in considerably faster reconstruction of high-resolution 3D images. Hierarchical octrees have been extensively used in a variety of image processing applications [14], [15], and these promising results in combination with an efficient approach to managing access to partitioned 3D spaces inspired us to use octrees with Morton keys for 3D FUS reconstruction. To our knowledge, this approach has not been previously used with 3D FUS.

This paper is organized as follows. In the next section, we describe the most common 3D FUS-based image reconstruction methods. Section III introduces hierarchical octrees and our methods incorporating Morton keys for neighbor search. Section IV presents our experiments with the octree-based methods and the results. To supplement our experiments, we also compare the time consumption and precision of our methods with those of a Bézier-based method with the best reported reconstruction times [16]. The final section presents our concluding remarks and discusses future work and potential upgrades.

II. BACKGROUND

The simplest conventional 3D FUS system consists of a 1D-array transducer with an attached tracking sensor. In this modality, the reconstruction of a 3D image starts by using the US transducer to scan several planes in the object of interest at different orientations and locations. This process produces a set $\mathcal{D}$ of conventional 2D US images containing the brightness values of all pixels and the data needed to map them to 3D space (i.e., position and orientation). This

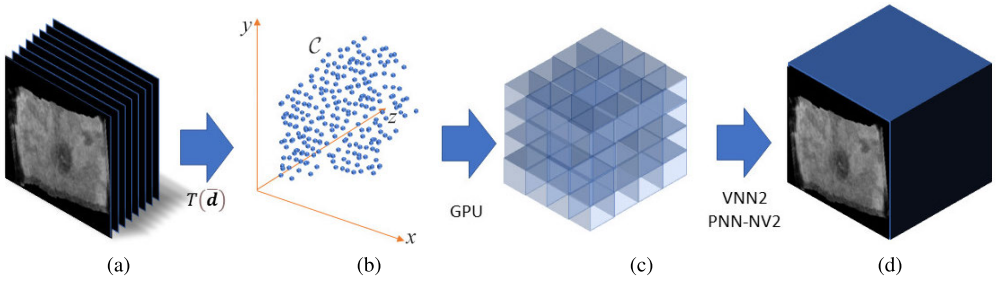


FIGURE 1. Pipeline of the proposed method for generating 3D images after acquiring (a) a set of 2D US images. (b) The pixels in all images are used to generate a bounded point cloud $\mathcal{C}$. (c) An octree with Morton keys completely encompassing the point cloud $\mathcal{C}$. (d) The final 3D image f produced by the octree-based image reconstruction method.

information is used to generate the final 3D US image. (In the following sections, when there is no potential for confusion, we refer to the brightness of a pixel or voxel simply as the value of that pixel or voxel.) To ensure that the method produces the highest quality 3D images, it is necessary to precisely know the position and orientation of the 2D US images with respect to the tracking sensor, which is an affine transformation including three rotations and translations. The parameters for this transformation are determined through a calibration process, and several calibration methods have been proposed in the literature. The mapping of all pixels in the images in $\mathcal{D}$ to 3D space yields what is commonly referred to as a point cloud, denoted by $\mathcal{C}$; the region occupied by $\mathcal{C}$ is then voxelized to produce the support vector V of the final 3D image f (i.e., $f : V \rightarrow \mathbb{R}$).

There are several methods for reconstructing 3D FUS images based on the pixels in the 2D US images. These methods can be classified into three main groups: *i*) pixel-based, *ii*) voxel-based, and *iii*) function-based methods. The first two approaches are based on searching values in neighborhoods, and the simplest of such approaches is a nearest-neighbor scheme [17]. Voxel-based methods first create an empty voxelized 3D image with the necessary dimensions to enclose $\mathcal{C}$; then, the voxels in this image are traversed and assigned a value based on the values of the neighboring pixels in $\mathcal{D}$. Conversely, pixel-based methods perform 3D FUS image reconstruction by first mapping all pixels in $\mathcal{D}$ to voxels in V and then assigning their corresponding values based on predetermined rules (i.e., bin filling); although the computing time for this step depends on the number of points in $\mathcal{C}$ (i.e., the number and size of the acquired 2D images), this process is generally fast. Subsequently, voxels that are not traversed during the bin-filling process are assigned values based on the values assigned to nearby voxels during the bin-filling process (i.e., hole filling); this process is usually computationally expensive because it depends on the number of voxels with unassigned brightness values and the method for computing their values [17].

The authors of [5] and [17] compared the quality of the reconstructed 3D images produced by pixel- and voxel-based methods and suggested that although basic methods, such as the voxel nearest neighbor (VNN) and pixel nearest neighbor (PNN) methods, achieve reasonable performance in terms of the computing time, the resulting 3D images have low-accuracy voxel values. In contrast, neighborhood search-based methods can produce 3D images with higher accuracy but with increased computational costs. Consequently, several methods to optimize the process of assigning the final voxel value by considering the values of nearby points have been proposed. For example, the authors of [18] proposed a method that identifies the two 2D US images in $\mathcal{D}$ that are closest to a voxel in V and assigns the final value based on the distance-weighted average of the values of the closest points in the images. Moreover, a common approach is to assign the final voxel value by considering the contributions of points within a vicinity (usually in the form of a kernel). This approach can improve the quality of the final 3D US image. However, an important limitation of this approach is the high computing cost of estimating the distance from a voxel to all the points within the vicinity. Fortunately, efficient data structures, such as octrees and Morton keys, can considerably reduce the computing cost.

The increasing availability of powerful computing hardware to efficiently perform parallel or distributed tasks and the development of advanced programming techniques has led to the development of GPU-based optimized methods for 3D FUS image reconstruction. For instance, the authors of [12] proposed two GPU-based reconstruction methods, a PNN method and a novel Bayesian method. The former can reconstruct 3D images with $161 \times 104 \times 232$ voxels in only 26.97 s, and the latter is 46.39 times faster than its CPU version and 2.86 times faster than the CPU version of the authors' own PNN implementation. Methods for intelligently representing data, such as octrees, and methods for parallelizing computing tasks have been widely adopted to accelerate neighbor searches in many applications involving large 3D datasets [14], [15]. The promising results produced by these kinds of approaches in other applications inspired us to develop an octree-based method to optimize the neighborhood search performed during 3D FUS image reconstruction. Furthermore, we used a neighborhood search-based approach because of its low computing time demands and the accurate

results reported when these methods have been used for previous 3D US image reconstruction tasks.

III. METHODS

A basic scheme demonstrating the process of reconstructing 3D US images with octrees, from the set of acquired 2D US images to the final 3D image f, is presented in Figure 1. As previously mentioned, the process begins with mapping the pixels in the acquired 2D US images in the set $\mathcal{D}$ to 3D space, resulting in the creation of the set $\mathcal{C}$. Notably, because our method maps the pixels to a point cloud, it is applicable with acquired 2D US images of different sizes. However, large size differences may lead to longer computation times and poorer accuracy. Then, the set V is created, which is associated with an octree with Morton keys. Finally, the 3D US image is generated based on the octree representation. Although many optimized 3D US image reconstruction methods that use neighbor search to compute the final brightness values in the 3D image have been developed, improved methods are needed to analyze larger volumes of interest. This is particularly important for neighbor search-based methods using a kernel, as optimizing the use of computing resources could lead to faster performance without reducing the image quality. Hence, we use octrees with Morton keys because this combination can improve the performance of neighborhood search-based methods, such as pixel- and voxel-based reconstruction methods.

A. OCTREES AND MORTON KEYS

We next introduce octrees and Morton keys, which are critical components of our proposed methods. An octree $\mathcal{O}$ is a tree-like data structure that is useful for efficiently partitioning 3D spaces into octants through recursive subdivision. To create such a representation based on a set such as $\mathcal{C}$, it is first necessary to obtain the minimal bounding box that surrounds all the points in the set. This region of space is represented by the tree's root node o_r. Then, the region of 3D space within the bounding box is subdivided into eight regions of equal size (octants), with each octant represented by a tree node o called a leaf o_f. Each region of 3D space within an octant (represented by a leaf node o_f) is then further subdivided into smaller octants, thus transforming the current leaf nodes o_f into internal nodes o and adding new leaf nodes to the octree. This process continues until a desired level of subdivision is reached (e.g., the subdivisions correspond to the voxels in V) or a maximum octree level is reached (i.e., the maximum number of subdivisions, or the resolution, of the octree). In our work, our datasets led us to consider octrees with a maximum of 10 levels, and these octrees are considered complete when each leaf node is associated with a voxel in V. Notably, once the octree is built with the above approach, some cells (i.e., voxels) associated with leaf nodes might be empty or contain more than one point $\boldsymbol{c} \in \mathcal{C}$.

In practice, there are several ways to implement an octree for 3D space partitioning, such as methods using registers with pointers to the tree nodes or Morton orders instead of pointers to the nodes. Although the former approach is straightforward to implement, this implementation uses significantly more computer memory than methods using the latter approach. In practice, trivial implementations of octrees are prohibitive in off-the-shelf GPUs. In contrast, approaches using Morton keys instead of pointers to address the octree nodes are at least as good at accessing the nodes but use considerably less memory, which potentially allows their use in GPUs.

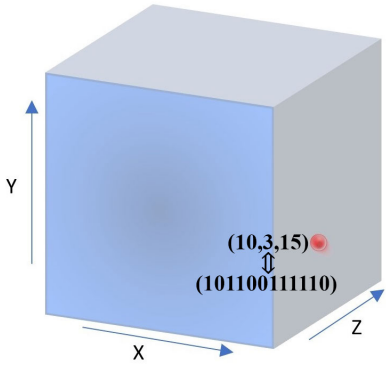


FIGURE 2. Example of the generation of the Morton key representation of a point c (red point) with coordinates of $(10, 3, 15)$; this representation is equal to $(1010, 0011, 1111)$ when using 4-bit coordinates. The Morton key is obtained by merging and reordering all of the binary digits to produce the sequence $\left(a_3^{(3)} a_2^{(3)} a_1^{(3)} a_3^{(2)} a_2^{(2)} a_1^{(2)} a_3^{(1)} a_2^{(1)} a_1^{(1)} a_3^{(0)} a_2^{(0)} a_1^{(0)}\right)$. Hence, the Morton key for the red point is (101100111110) or $0\times$B3E in hexadecimal notation.

A Morton order is a mapping from an n-dimensional space to a linear list of numbers. In this mapping, coordinates close to each other in the n-dimensional space have Morton numbers, or keys, that are also close to each other. In our application, a Morton key $\mathcal{M}$ is a sequence of binary numbers whose values are assigned using the coordinates of the point to be represented. A point $\boldsymbol{u} \in \mathbb{Z}_+^3$ can be represented by a 3-tuple (u_1, u_2, u_3), where $u_j \in \mathbb{Z}_+$ for $1 \leq j \leq 3$; thus, every coordinate u_j can be represented by a sequence of binary numbers $a_j^{(m-1)}, \ldots, a_j^{(0)}$, where $a_j^{(l)} \in \{0, 1\}$ for $1 \leq l \leq m$. As a result, the Morton key $\mathcal{M}_{\boldsymbol{u}}$ corresponding to a position $\boldsymbol{u} \in \mathbb{Z}_+^3$ can be obtained by merging and interleaving all the binary digits of its coordinates as follows: $a_3^{(m-1)}\ a_2^{(m-1)}\ a_1^{(m-1)} \ldots a_3^{(l)}\ a_2^{(l)}\ a_1^{(l)} \ldots a_3^{(0)}\ a_2^{(0)}\ a_1^{(0)}$ (see Figure 2). This coding scheme applies to any voxel $\boldsymbol{v} \in V$ with integer coordinates (v_1, v_2, v_3). However, for points $\boldsymbol{c} \in \mathcal{C}$, Morton keys cannot be used directly. Hence, to use Morton keys for all types of coordinates involved in the process of reconstructing 3D US images, we first normalize the coordinates based on the size of the octree in every dimension and translate all points in $\mathcal{C}$ to ensure that all points are in the positive octant. Additionally, to ensure that the method can represent any point with Morton keys, we use only two decimal places to represent real numbers, and all coordinates are scaled by a factor of one hundred. In the present implementation, we use 10 bits for every coordinate (i.e., m is equal to 10), which allows for a maximum value of 1024. With this encoding,

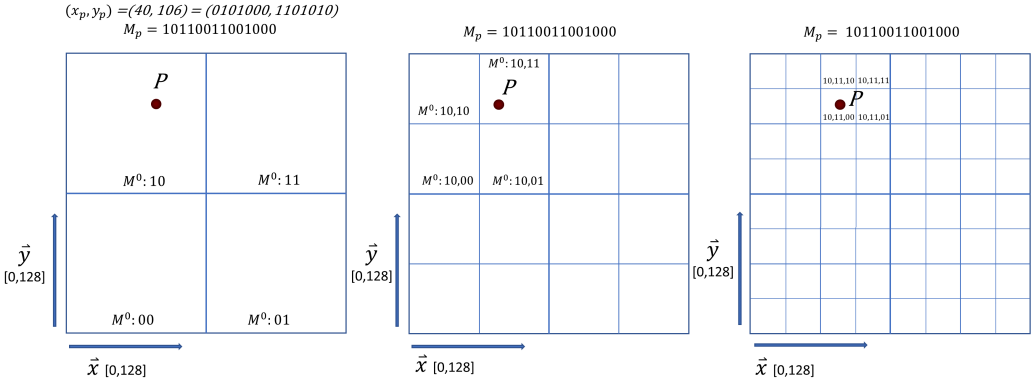


FIGURE 3. Usage of Morton keys as filters. From left to right, each frame shows how the 2D space is progressively represented by a quadtree (a 2D equivalent of an octree), from the first-level division to the level at which the point p is represented by a single node. In addition, we show how the Morton keys $\mathcal{M}^o$ are assigned to the subdivisions in each level, demonstrating how the Morton keys of lower-level nodes serve as filters for their offspring nodes.

each Morton key uses 30 bits, which we store as 32-bit integers, leaving two bits unused.

When Morton keys are used in an octree, each node is assigned a unique code (i.e., a location code). For a node $o \in \mathcal{O}$, its key $\mathcal{M}^o$ is a sequence of digits tracing a path from the root node o_r to o; additionally, this type of key identifies the set of branches that must be traversed on the given path [19]. Because the leaf nodes in an octree store positions in space related to the voxels in V, we can refer to the leaf nodes by using the coordinates of the corresponding voxels (i.e., in this representation, a voxel $\boldsymbol{v}$ refers to both the corresponding voxel and its associated leaf node).

Another advantage of using octrees with Morton keys is that the Morton keys are arranged hierarchically in the nodes based on their importance; this ranking improves computational efficiency by allowing regions to be quickly filtered out when conducting searches in the octree. Each node $o \in \mathcal{O}$ has a Morton key serving as both an identifier and a filter. In this representation, the level occupied by any internal node $o \in \mathcal{O}$ determines the region it represents as follows: the Morton keys for the ith level nodes are created using the $3 \times i$ most significant bits of their coordinates' binary representations, with one bit used for each axis. Thus, the bits associated with the Morton key of an internal node o are shared by the Morton keys associated with all descendant nodes of o. Moreover, the Morton keys for each leaf node $o_f \in \mathcal{O}$ include all the bits of the integer coordinates. Hence, this kind of Morton key and its successive keys can be used as masks to efficiently filter nodes (i.e., regions) and points (see Figure 3).

One final advantage of using an octree with Morton keys is that the whole tree can be stored as a vector V (a one-dimensional array in memory), in which the octree's root node is the first element in the array and the nodes in the following levels are stored sequentially. The nodes in the ith level are stored sequentially based on their $3 \times i$ most significant bits (e.g., in level 1, three bits are used to store eight positions, and in level 2, six bits are used to store 64 positions).

Although the usage of an octree in combination with Morton keys considerably improves the speed of searches and assignments, the speed even be improved further by using a list $\mathcal{L}$ to store all the points within the 3D region represented by an octree $\mathcal{O}$. This list can be created in either descending or ascending order based on the Morton keys of the points. Additionally, for every node $o \in \mathcal{O}$, our method stores an extra integer index that refers to the first point $\boldsymbol{c} \in \mathcal{C}$ in $\mathcal{L}$ that falls within the 3D region represented by o. Thus, the Morton key-based sorting method guarantees that points in $\mathcal{L}$ stored after $\boldsymbol{c}$ are also located within the region represented by node o. Thus, it is sufficient to use part of a Morton key to verify whether a point $\boldsymbol{c}$ is within a certain region of space. As a result, our method can quickly search for points inside the regions covered by $\mathcal{O}$. To our knowledge, the use of such an ordered list has not been proposed elsewhere in the literature. In our present implementation, we use 3 Morton key bits for each level in the octree, starting with the most important bit (i.e., for the first level in the octree, we use the three most significant bits; for the second level, we use the six most significant bits). Because each Morton key contains the three-dimensional coordinates associated with the center of the corresponding voxel $\boldsymbol{v}$, the search for the neighbors of $\boldsymbol{v}$ becomes a search for their associated nodes, which can be quickly performed with binary operations based on their Morton keys, following by accessing the nodes from the associated list $\mathcal{L}$.

Finding all the neighbors of a voxel within a given radius is an integral step in many existing 3D FUS image reconstruction methods. When using an octree representation, the search for all neighbors of a position $\boldsymbol{u}$ in 3D space produces the set

$$\mathcal{Q}_{\boldsymbol{u}} = \left\{ o_f \,|\, \mathcal{V}_{\boldsymbol{v}(o_f)} \cap \mathcal{S}\,(\boldsymbol{u}, \rho) \neq \emptyset \right\}, \tag{1}$$

where $\mathcal{S}$ is a kernel centered on $\boldsymbol{u}$ that tightly fits within the ρ-radius sphere and $\mathcal{V}_{\boldsymbol{v}(o_f)}$ is the Voronoi neighborhood of $\boldsymbol{v}$ in V associated with the leaf node o_f. The kernel can have various shapes, including cubic or spherical, as long

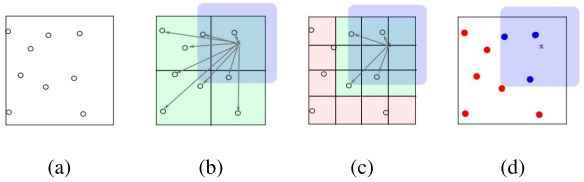


FIGURE 4. Neighborhood search using octrees. The distances to objects (circles) within the overlapping octants (green) must be computed, while the distances to objects in nonoverlapping octants (red) can be excluded. (a) The undivided original region containing objects (i.e., only the root node). (b) Neighborhood search in the first-level division, where all octants overlap with the kernel (blue). (c) Second-level division; some octants do not overlap. (d) Objects inside (blue) and outside (red) the kernel.

as it fits inside the sphere. After obtaining the set $\mathcal{Q}_{\boldsymbol{u}}$ of all neighboring voxels to $\boldsymbol{u}$, the method searches for all the pixels mapped to these voxels. This process creates the following set for each leaf node $o_f \in \mathcal{Q}_{\boldsymbol{u}}$:

$$\mathcal{P}_{\boldsymbol{u},o_f} = \left\{ \boldsymbol{c} \in \mathcal{C} | \boldsymbol{c} \cap \mathcal{V}_{\boldsymbol{v}(o_f)} \neq \emptyset \text{ and } \|\boldsymbol{u} - \boldsymbol{c}\| \leq \rho \right\}. \quad (2)$$

This approach restricts the search to points in the overlapping leaf nodes $o_f \in \mathcal{Q}_{\boldsymbol{u}}$; thus, large subsets of points that are irrelevant to the query are discarded [14] (see Figure 4 for an illustration of this neighbor search process).

As previously mentioned, an important limitation of octrees is the potentially large amount of memory required to store them, even when using Morton keys, because their size increases exponentially as more subdivisions are added. As a result, a prohibitive amount of computer memory may be required to store complete octrees representing large 3D images. This issue can be addressed by applying the sparse-voxel octree method, in which only nodes with information are represented, leaving the remaining nodes undefined and thus not stored in memory [20].

Currently, modern computer hardware is designed with parallel architectures with multiple processors or cores. Thus, it would be desirable to exploit such an architecture and use multiple processors or cores simultaneously to reconstruct 3D US images. GPUs have become an alternative for parallel computing because they have thousands of efficient cores designed for handling multiple tasks simultaneously and can perform many parallel tasks. Nvidia® GPUs have become widely used, and Nvidia® also offers CUDA®, a platform for general parallel computing. Therefore, we decided to use this environment to implement our methods; however, it is worth noting that our methods could be implemented on other efficient and large-scale parallel computing platforms for parallel data computations. The CUDA® architecture uses a multithreaded programming model in which parallel tasks in an application are executed on hardware as kernels; in turn, several threads execute each kernel. Multithreading allows the efficient generation and assignment of Morton keys by quickly traversing all nodes in the octree. By using a vector V of nodes, CUDA® kernels can be used to verify all nodes, and their positions in the vector can be used to determine their depth and, therefore, their Morton keys. Furthermore, the use of Morton keys enables not only independent navigation through nodes but also independent processing of the information associated with these nodes. In addition, we can exploit threads in CUDA® to independently process the information associated with these nodes, and these threads do not need to wait for other threads to finish. Importantly, our implementations in CUDA® mainly exploit the properties of Morton keys and the spatial relationships they have with both nodes in $\mathcal{O}$ and points in $\mathcal{C}$. The greater the extent to which these relationships are used to reconstruct a 3D image, the more fully neighbor search-based reconstruction methods can exploit our proposed octree- and Morton key-based strategies.

At this point, it is worth noting that we are not simply using faster and more powerful hardware to solve the 3D image reconstruction problem; such naive attempts would not take full advantage of the hardware capabilities (e.g., failure to make full use of independent parallel tasks would lead to synchronization overhead). Instead, our method parallelizes many octree-related tasks by following three main strategies. First, our method stores all nodes in an octree $\mathcal{O}$ in an ordered linear array $\mathcal{L}$ based on their Morton keys. Therefore, the properties of any node can be accessed and modified independently, regardless of the level of the node. In addition, for any node, it is straightforward to obtain its parent and sibling nodes by calculating their respective Morton keys. The properties of any leaf node can also be accessed or modified by converting the Morton key representation used in the octree into the corresponding node index representation used in the linear array and vice versa. Thus, pointers and other complex data structures are not required to independently process the final elements in the linear array (those representing the highest levels of the octree).

B. 3D IMAGE RECONSTRUCTION

Once the full octree with Morton keys has been constructed, we apply a reconstruction method based on neighbor search to produce the final 3D US image. In our work, because methods based on neighbor search tend to produce high-quality 3D US images, we implement a voxel-based method (referred to as the voxel vicinity neighbor with octree and Morton (VVN-OM) method) and a pixel-based method (referred to as the pixel vicinity neighbor with octree and Morton (PVN-OM) method) using a neighbor search-based strategy to assign the final voxel values to the 3D US images. Moreover, we implement our proposed scheme with octrees and Morton keys in both methods because there was no *a priori* evidence that one of these methods performed better than the other when optimized with such an approach.

Our VVN-OM method traverses every leaf node o_f in $\mathcal{O}$ to compute the final value of its associated voxel $\boldsymbol{v}\left(o_f\right)$; this value is obtained by first searching for neighboring points in $\mathcal{C}$ using a kernel $\mathcal{S}_{\boldsymbol{v},\rho}$ that is centered on the voxel $\boldsymbol{v}$ (i.e., $\left\{\boldsymbol{c} \in \mathcal{C} | \boldsymbol{c} \cap \mathcal{S}_{\boldsymbol{v},\rho} \neq \emptyset\right\}$). This search starts with the kernel $\mathcal{S}_{\boldsymbol{v},\rho}$ whose size ρ covers only the voxels adjacent to $\boldsymbol{v}$; if there are no points in $\mathcal{C}$ within the kernel, the size ρ is

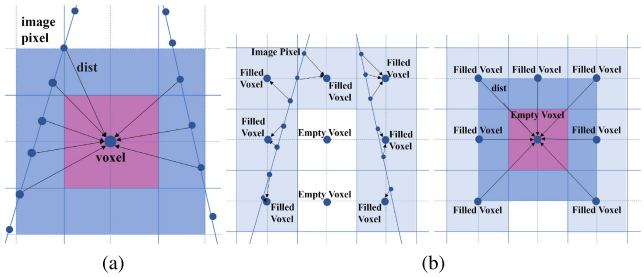


FIGURE 5. (a) In the VVN-OM method, a kernel $\mathcal{S}$ (pink) is centered on a voxel $v \in V$ that does not have any pixel in $\mathcal{C}$ within its Voronoi neighborhood. The size of this kernel is increased (blue) until it intersects with pixels in $\mathcal{C}$; the values of these pixels contribute to the value of v. (b) In the PVN-OM method, first (left), the values of every mapped pixel $c \in \mathcal{C}$ are assigned to their closest voxel $v \in V$ (bin filling). Then (right), every $v \in V$ not previously visited is assigned a value as a distance-weighted combination of values from all voxels that have already been filled and intersect with the kernel $\mathcal{S}$ centered on v (hole filling).

increased, and the search process is restricted to the newly added voxels (see Figure 5(a)). The search for voxels within the $\boldsymbol{v}$-centered kernel $\mathcal{S}_{\boldsymbol{v},\rho}$ is expedited by the use of Morton keys because all voxels in this region should have similar keys; thus, the neighbors of a voxel $\boldsymbol{v}$ (i.e., leaf nodes) can be quickly identified by applying binary operations to the Morton key of $\boldsymbol{v}$. This strategy uses the index of the first point in $\mathcal{L}$ that falls within the region, and using the Morton keys is straightforward method to identify the remaining points in the region. Furthermore, the points $\boldsymbol{c} \in \mathcal{C}$ within this region of space should have similar Morton keys, making them relatively easy to find. Once the search for neighboring points is completed (see (1)), the final value is calculated as [17]:

$$\phi(\boldsymbol{v}) = \frac{1}{|\mathcal{S}_{\boldsymbol{v},\rho}|} \sum_{\boldsymbol{u} \in \mathcal{S}_{\boldsymbol{v},\rho}} g(\boldsymbol{u})\, e^{-\left(\frac{\|\boldsymbol{u}-\boldsymbol{v}\|}{\sigma}\right)^2}, \tag{3}$$

where $|\mathcal{S}_{\boldsymbol{v},\rho}|$ is the number of points in $\mathcal{C}$ discovered within the kernel $\mathcal{S}_{\boldsymbol{v},\rho}$, the function g yields the brightness value of the corresponding point $\boldsymbol{u} \in \mathcal{C}$, $\|\bullet\|$ is the l_2-norm, and σ is the longest distance between the center of $\mathcal{S}_{\boldsymbol{v},\rho}$ and the farthest valid position.

In contrast, our PVN-OM method starts by traversing every point $\boldsymbol{c} \in \mathcal{C}$ and assigning their values to their closest voxels (i.e., the bin-filling process). Specifically, in this stage, the method assigns the value of a pixel $\boldsymbol{c} \in \mathcal{C}$ to a voxel $\boldsymbol{v} \in V$ when the point $\boldsymbol{c}$ is within the Voronoi neighborhood $\mathcal{V}_{\boldsymbol{v}}$ of the voxel $\boldsymbol{v} \in V$. Through parallelization, this process can be efficiently carried out by independently processing the Morton key of every internal node $o \in \mathcal{O}$ as a mask. This process also helps determine the corresponding associated leaf nodes. When there is more than one point $\boldsymbol{c} \in \mathcal{C}$ inside $\mathcal{V}_{\boldsymbol{v}}$ (i.e., $|\{\boldsymbol{c} \in \mathcal{C} | \mathcal{V}_{\boldsymbol{v}} \cap \boldsymbol{c} \neq \emptyset\}| > 1$), the associated values of all points are averaged to determine the final value of $\boldsymbol{v}$. Following the bin-filling process, all nonvisited voxels (i.e., holes) are visited, and their values are computed following a process similar to the one used in our VVN-OM method using Equation (3). However, instead of identifying points $\boldsymbol{u} \in \mathcal{C}$ intersecting with the kernel $\mathcal{S}_{\boldsymbol{v},\rho}$, this method searches for voxels $\boldsymbol{v}\,(o_f)$ for $o_f \in \mathcal{O}$ whose values were assigned during the bin-filling process that intersect with $\mathcal{S}_{\boldsymbol{v},\rho}$. Crucially, any voxel whose value is assigned during the hole-filling process is considered a synthetic node and is excluded from the remaining process (see Figure 5(b)).

Finally, because all points in $\mathcal{C}$ are also stored in a sorted linear array based on the Morton keys, it is easy to move back and forth between the point space and the voxel space. In our CUDA® implementation, this design allows us to rapidly transfer information between $\mathcal{C}$ and V, fill the octree $\mathcal{O}$, and perform other operations in our methods. In our implementations of the PVN-OM and VVN-OM methods, we use 4 CUDA® kernels to prevent conflicts among the different stages of the algorithms. One of the kernels is responsible for creating the octree by processing every node in the linear array. During this process, the necessary Morton keys are also created. The second kernel processes all the points and assigns and sorts their Morton keys. Finally, the bin- and hole-filling processes are performed by the third and fourth kernels, respectively. The VVN-OM method uses a single CUDA® kernel employing one thread per leaf node $o_f \in \mathcal{O}$. Furthermore, every thread uses the Morton key of o_f to search for the Morton keys of its neighboring nodes; these keys are then used to filter points within the neighborhood. The PVN-OM method uses a single CUDA® kernel employing one thread per point $\boldsymbol{u} \in \mathcal{C}$. A thread uses the Morton key of a point $\boldsymbol{u}$ to obtain its position in the vector V of nodes. This approach straightforwardly determines the leaf node o_f where $\boldsymbol{u}$ is located and where its information should be processed. Notably, in the PVN-OM method, atomic operations are used in CUDA® to ensure that different threads do not concurrently access the same node. In both methods, a different CUDA® kernel is response for the hole-filling process, and one thread is used for each leaf node $o_f \in \mathcal{O}$.

IV. EXPERIMENTS AND RESULTS

In this section, we report on the experiments designed to test the performance of our VVN-OM and PVN-OM methods. We developed software implementing these methods on GPUs. For comparison purposes, we also implemented versions of these methods without using octrees or Morton keys on GPUs; we refer to these versions as VVN and PVN, respectively. These implementations use CUDA® to efficiently exploit the resources of the Nvidia® hardware in our computer systems. As mentioned in INTRODUCTION, there is usually a tradeoff between speed and accuracy when reconstructing 3D US images; hence, we also present an experiment designed to measure the accuracy of our two methods. Finally, we present an experiment to compare our methods to a recently proposed function-based method.

For our experiments, we use two desktop systems using a single Intel® Core™ i7 processor with 4 cores, running at 4.2 GHz, with 16 GiB of RAM. One of these systems uses a GeForce® GTX 1080 as its GPU, powered by the Pascal™ architecture, with 6 GiB of memory, while the other uses a

TABLE 1. Times needed to reconstruct the 3D US images of different dimensions based on the 2D US images with dimensions 387 × 400 px using our pixel- and voxel-based methods with and without octrees and Morton keys.

# 2D Images	3D Image (voxels)	Time (seconds)			
		PVN-OM	VVN-OM	PVN	VVN
131	128×128×128	0.72	1.07	4.98	2.71
	256×256×256	7.86	8.71	24.47	16.22
	512×512×512	79.79	86.28	121.71	239.55
66	128×128×128	0.54	0.84	5.07	1.53
	256×256×256	6.19	6.93	26.07	17.15
	512×512×512	70.57	70.84	107.40	185.89
51	128×128×128	0.44	0.80	4.56	1.02
	256×256×256	5.22	6.36	23.89	12.95
	512×512×512	71.16	67.64	104.48	150.23

GeForce® RTX 2060, powered by the Turing™ architecture, also with 6 GiB of memory. Importantly, all our experiments were executed using only a single CPU core.

To produce the necessary data for the experiments, we employed a 3D FUS system using an optical tracker (Polaris Spectra by Northern Digital Inc., Canada) and a 2D US system (Aloka SSD-1000 Diagnostic Ultrasound System by ALOKA Co., Ltd., U.S.A.) to generate experimental data. As mentioned in Section II, the system must be calibrated to obtain a set $\mathcal{C}$ of points. For the spatial calibration of our FUS system, we used a crosswire phantom (a tray filled with water with two submerged wires crossing at their center). We selected this phantom due to the simplicity and ease of construction. To calibrate our system, we followed the procedure described in [21]; a 1D-array US transducer was used to acquire a set of 2D images with different orientations in the region where the wires cross to compute values for the affine transformation from the scanning plane to the dynamic reference frame.

The above equipment was used to scan two phantoms. The first phantom was a polyvinyl alcohol (PVA) phantom simulating a breast with a lesion. The second phantom was a General-Purpose Ultrasound Phantom Model 044 manufactured by CIRS, Inc., U.S.A.; this phantom is a cuboid block measuring 18 × 13×20 cm^3, with several short cylinders arranged in two planes, and is designed to simulate the US properties of human liver tissue. However, we used only one of these planes for our experiments.

To produce the necessary 2D US data for our experiments, we acquired several US images, all 387 × 400 pixels (px) in size, of the aforementioned phantoms using the Aloka system at a frequency of 7 MHz. For the PVA phantom, we obtained a single dataset containing 131 images. For the CIRS phantom (GPUP), we acquired two datasets. The first dataset contained 66 images of the region of the plane with the 1.5-mm-diameter cylinders, covering only 6 of the cylinders. The second dataset contained 51 images of the region of the plane with the 3-mm-diameter cylinders, covering only 9 of the cylinders.

A. RECONSTRUCTION TIME AND MEMORY USAGE

We assessed the computational times of our proposed PVN-OM and VVN-OM methods by measuring the time they needed to reconstruct 3D images with dimensions of 128 × 128×128, 256 × 256×256, and 512 × 512×512 voxels. Then, we compared their execution times with those of the octree-less PVN and VVN methods. To run these experiments, we used the computer system with the GTX 1080 GPU and the 2D US images obtained from the CIRS and PVA phantoms as input for our methods. In Table 1, we show the times needed by these methods to reconstruct the various 3D images. To visually evaluate these results, Figure 6 shows an example of a reconstructed 3D image of the PVA phantom with 256 × 256×256 voxels.

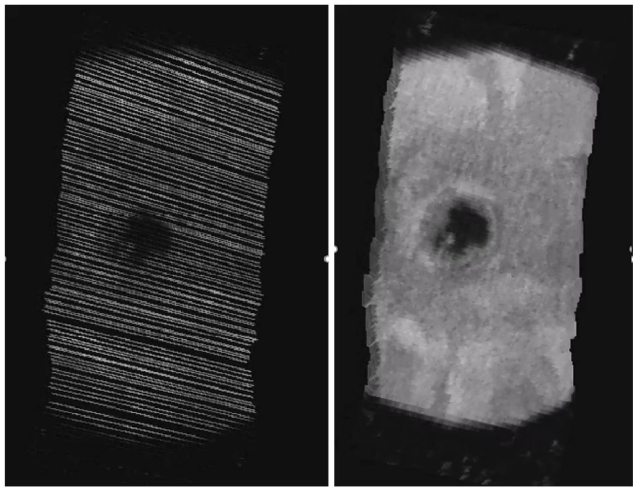


FIGURE 6. Maximum intensity projection on the axial plane of a 3D image with dimensions of 256 × 256×256 voxels produced using the PVA phantom dataset. The voxel values in the 3D image are assigned by (left) using only the 2D images acquired with a 1D-array transducer and (right) applying the VVN-OM method to the acquired 2D images.

To evaluate the sppedup achieved by using octrees and Morton keys, we used the following measure:

$$SpeedUp = \frac{T_n}{T_o}, \tag{4}$$

where T_n is the time used by an octree-less method and T_o is the time used by a method using octrees with Morton keys. The results in Table 1 show that the best acceleration was 10.28×, which was obtained with the PVN-OM method when the dataset with 51 2D US images was used to reconstruct a 3D US image with 128 × 128×128 voxels. The worst acceleration was 1.25×, which was obtained with the VVN-OM method when the dataset with 51 2D US images was used to reconstruct a 3D US image with 128 × 128×128 voxels.

It is worth noting that Huang and Zeng [5] reported that the fastest sequential implementation of the pixel-based method using nearest-neighbor searches required 0.031 s to process 302 × 268 px 2D US images to reconstruct a 3D US image with 90 × 81×192 voxels. In comparison, our PVN-OM method needed only 0.005 s for every 387 × 400 px 2D US image to reconstruct a 3D US image with 128 × 128×128 voxels (approximately 40% more voxels).

Furthermore, we note that one of the most computationally demanding tasks is assigning the points in $\mathcal{C}$ to each voxel in

TABLE 2. Memory in megabytes used to reconstruct 3D US images of different dimensions using our PVN-OM and VVN-OM methods.

2D Images (387×400 px)	3D Image (voxels)	Memory (MiB) PVN-OM	VVN-OM
131	128×128×128	1,390	1,538
	256×256×256	1,614	1,762
	512×512×512	3,406	3,554
66	128×128×128	1,278	1,388
	256×256×256	1,502	1,612
	512×512×512	3,294	3,404
51	128×128×128	1,196	1,280
	256×256×256	1,420	1,504
	512×512×512	3,212	3,296

TABLE 3. Mean measured values of the diameter and separation of the cylinders in the reconstructed 3D images of the CIRS phantom (GPUP).

	Cylinder Set 1 (mm)		Cylinder Set 2 (mm)	
	Diameter	Separation	Diameter	Separation
VVN-OM	1.49±0.05	12.60±0.13	2.95±0.08	12.60±0.10
PVN-OM	1.54±0.03	12.60±0.10	3.14±0.02	12.60±0.19

V. When using our approach with octrees and Morton keys, this process is not performed as a separate task because of the spatial relationships among the Morton keys.

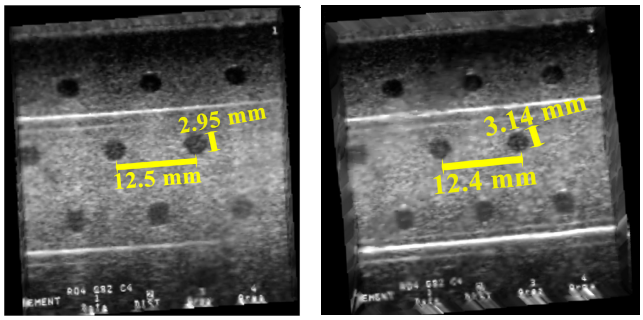
In many computer applications, there is a tradeoff between memory usage and computing time. Thus, we also determined the memory usage of our two methods when reconstructing the 3D US images, and the results are shown in Table 2. In our proposed methods, the number of octree nodes increases exponentially by 8^n, where n is the depth of the octree. In our implementations, every octree node uses 0.015625 KiB, and an 8-level octree (256 × 256×256 cells) requires 292.57 MiB of storage. When the octree's depth is increased by one, the number of additional cells is equal to 512 × 512×512, and the required memory is increased to 2.28 GiB. Current off-the-shelf GPUs can easily handle such memory demands.

B. RESOLUTION ASSESSMENT

We also assessed our methods in terms of the resolution they can achieve when reconstructing various 3D images using the datasets acquired in the GPUP planes. These 3D images had dimensions of $64\times64\times64$, $128\times128\times128$, $256\times256\times256$, and $512\times512\times512$ voxels. During the reconstruction process, a maximum kernel size of $9\times9\times9$ voxels was used for both methods. After obtaining the 3D images, we manually measured the diameters of all small cylinders and their separation distances. In Table 3, we present the average values of the cylinder diameters and their standard deviations for the different reconstructed 3D images. Figure 7 shows two reconstructed $256\times256\times256$.3D images of the region where the 3-mm cylinders are located.

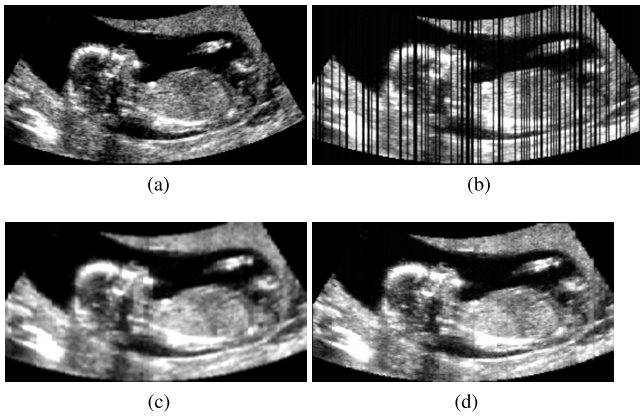
C. ERROR ASSESSMENT

Finally, we assessed the accuracy of our methods by measuring how well they reconstructed the values in several sets of 3D US images, which were taken as ground truth. For each



(a) (b)

FIGURE 7. Reconstructed 3D images with dimensions of 256 × 256 × 256 voxels in the region of the CIRS phantom in which the 3-mm cylinders are located; note the straight lines delimiting the region containing the cylinders. The reconstructions were obtained with the (a) VVN-OM and (b) PVN-OM methods. The yellow lines and text illustrate the acquired dimensional measurements and their corresponding measured values.



(a) (b) (c) (d)

FIGURE 8. (a) The sagittal plane in a ground-truth 3D fetal image. (b) The same plane in a 3D image with 30% of the voxels removed. (c, d) The corresponding planes in the reconstructed 3D images obtained with the (c) PVN-OM and (d) VVN-OM methods.

of the 3D US images, we selected several slices perpendicular to the $\overline{xy}$, $\overline{xz}$, or $\overline{yz}$ planes. We followed this approach because it resembles the method for generating datasets of 2D US images using freehand probes with uniform acquisition schemes. Moreover, to better simulate a realistic scenario, we created gaps in the information by randomly removing different numbers of 2D US images from the acquired dataset. This methodology allowed us to assess the error associated with the reconstruction algorithms while canceling out any errors introduced during the acquisition of the 2D US images. Another advantage of this approach is that these robustness tests allow a common frame for comparison with other 3D US reconstruction methods.

We measured the difference between the values in a reconstructed 3D image and the corresponding ground-truth values by calculating the following normalized mean square error (MSE):

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(f_r\left(\boldsymbol{c}_i\right) - f_b\left(\boldsymbol{c}_i\right)\right)^2, \tag{5}$$

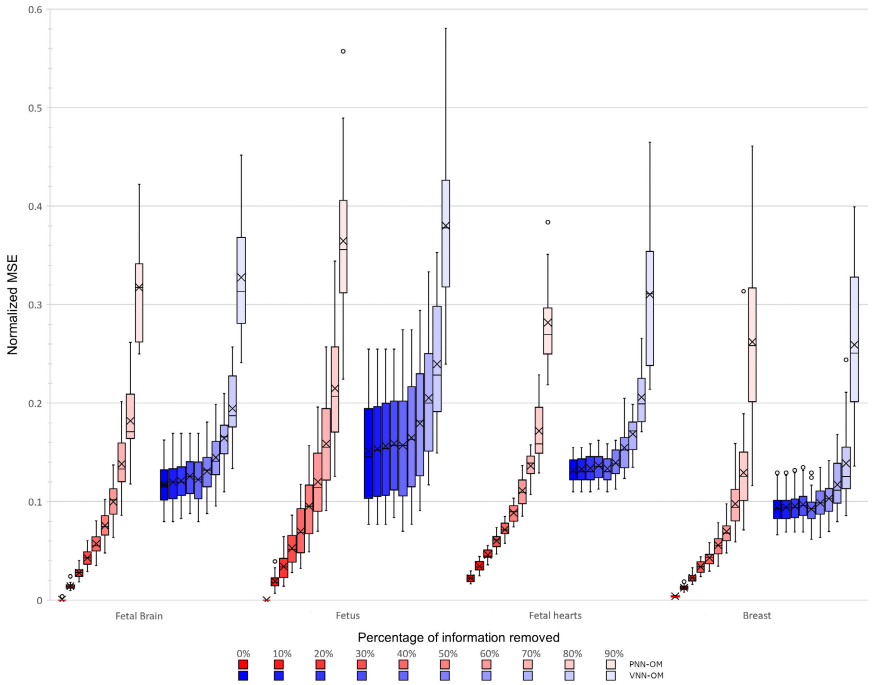


FIGURE 9. Increase in $\overline{\text{MSE}}$ as a function of the amount of information eliminated from the dataset (represented by different shades of the corresponding hue) when producing 3D US images with the (red) PVN-OM and (blue) VVN-OM methods. For these tests, we used the Fetal Brains, Fetuses, Fetal Hearts, and Breast datasets.

where $f_r(\boldsymbol{c}_i)$ and $f_b(\boldsymbol{c}_i)$ are the values for the ith voxel $\boldsymbol{c}$ in the reconstructed and ground-truth 3D US images, respectively, and N is the total number of voxels. For a given ground-truth 3D US image, we created several datasets of 2D US images with different amounts of deleted information, ranging from 100% exclusion to no information removed. The error produced with the zero-information-removed dataset was used to normalize the errors obtained in reconstructing the remaining 3D US images. We denote this baseline error as MSE_M; hence, the final reported error for a reconstructed 3D US image is the ratio between its MSE and MSE_M, and we denote this ratio as $\overline{\text{MSE}}$.

For this task, we obtained 92 ground-truth 3D US images of 20 brains, 61 fetuses, and 11 fetal hearts; these images were generated with a General Electric Voluson 3D US system using a frequency ranging from 2 to 7 MHz. We also obtained 31 ground-truth 3D US images representing breasts with different characteristics. These images are available in a public database coauthored by the Department of Radiology at Tahammasat University and the Queen Sirikit Center of Breast Cancer of Thailand [22]. In Figure 8, we present several sagittal planes in the fetal images illustrating the error measurement process.

For this assessment, we ran our VVN-OM and PVN-OM implementations on the computer system with the RTX 2060 GPU, and we report the $\overline{\text{MSE}}$ values and standard deviations for all 3D US images reconstructed with these methods in Figure 9.

The question remains as to whether the utilization of octrees and Morton keys introduces errors in the pixel- and voxel-based methods. To investigate these potential errors, we compared the PVN and VVN methods with their corresponding versions using octrees and Morton keys. In Figure 10, we compare the $\overline{\text{MSE}}$ values obtained by the PVN and PVN-OM methods, and the VVN and VVN-OM methods show similar behaviors. The average difference between the errors produced by both methods is 0.00089, with the PVN-OM method having a smaller error; hence, we can consider these values negligible. The errors of both methods are mainly due to precision in computing the distances and neighbors; in particular, the method using octrees and Morton keys quantifies distances. These results suggest that our methods using octrees and Morton keys do not introduce significant errors during the reconstruction process. We emphasize that the aim of these experiments was to have a comparison excluding the noise introduced in acquiring the US images to enable a fair comparison between methods with and without octrees.

D. COMPARISON WITH A BÉZIER-BASED METHOD

Although our proposed approach is limited to optimizing neighbor search-based reconstruction methods, we complement our assessment with a comparison with an algorithm using Bézier interpolation [16]. In Table 4, we compare the time performance of our two methods with that of the Bézier-based method when reconstructing a $256\times256\times256$.3D US

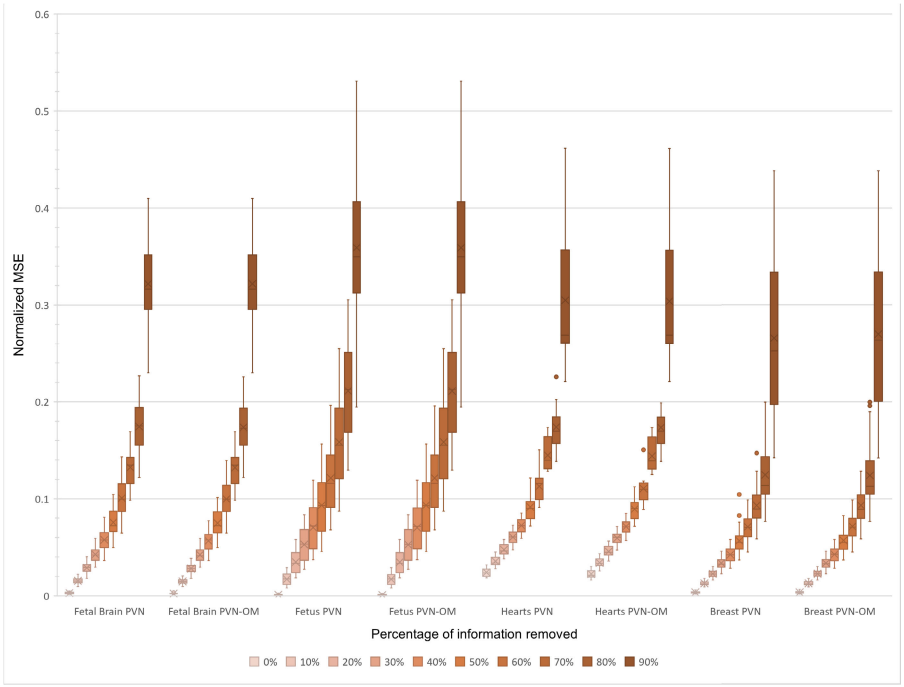


FIGURE 10. Comparison between the increase in $\overline{\text{MSE}}$ as a function of the amount of information eliminated from a dataset (represented by different shades of the corresponding hue) when producing 3D US images with the PVN and PVN-OM methods. For these tests, we used the Fetal Brains, Fetuses, Fetal Hearts, and Breast datasets.

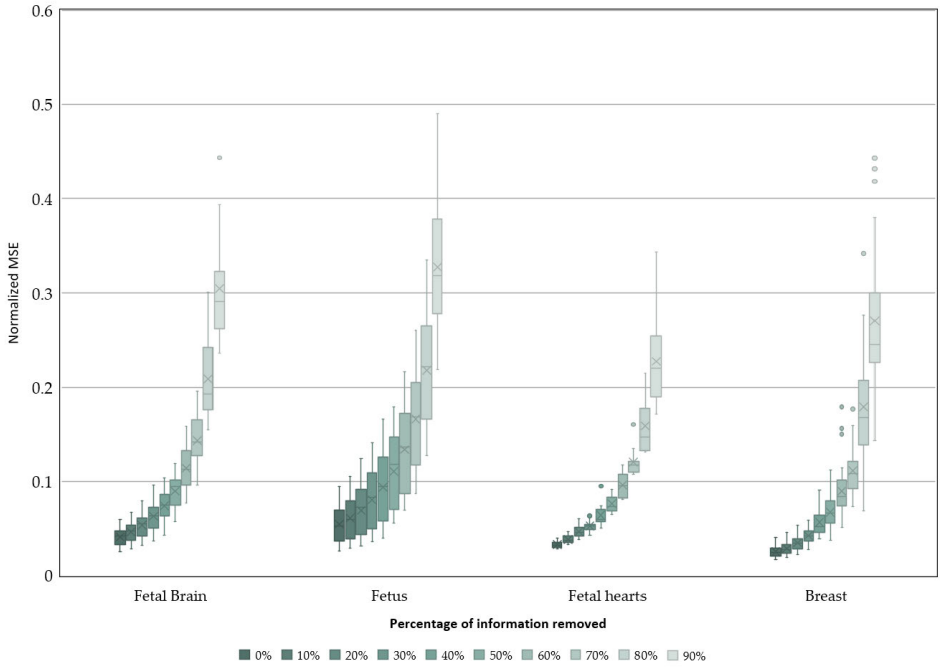


FIGURE 11. The $\overline{\text{MSE}}$ values for the Bézier-based reconstruction algorithm when different amounts of information were eliminated from the dataset, represented by different shades. We tested this algorithm using the same datasets as in Figure 9.

image. For this test, we used the CIRS phantom (GPUP) dataset. Additionally, we measured the MSE results for the Bézier-based method based on the same datasets used to assess our methods, and we present these results in Figure 11.

TABLE 4. Comparison of the time needed to reconstruct a 3D US image with dimensions of 256 × 256×256 using a Bézier-based method and our PVN-OM and VVN-OM methods for datasets with different numbers of 2D US images.

Images (387×400 px)	Time (seconds)		
	Bézier	PVN-OM	VVN-OM
131	0.20	7.86	8.71
66	0.08	6.19	6.93
51	0.07	5.22	6.36

According to the results in Table 4, the Bézier-based method achieves a better time performance; however, this reduced computing time is balanced by the increased error compared with our PVN-OM method. Conversely, our VVN-OM method always performs worse than the Bézier-based method. In the Bézier-based method, four consecutive 2D US images are used for interpolation [16]. Four pixels with the same position in the corresponding consecutive images are used as the control points for the 3rd-degree Bézier curve. Then, this curve is used to compute the final values of the voxels intersecting the curve. This method achieves excellent performance when the input images have the same dimensions and are not too separated. However, the Bézier-based method is sensitive to the orientations and distances among the acquired 2D US images.

The Bézier-based method requires that the acquired images do not deviate much from a given orientation to yield a satisfactory 3D US image, whereas our methods do not have such a restriction, providing more leeway in the acquisition process. The Bézier-based method also encounters problems when the input images cross with the underlying Bézier spline because ensuring the continuity of the curve is difficult without increasing the number of control points; images crossing each other are not unlikely in 3D FUS because the orientation of the scanned 2D images depends on the dexterity of the technician acquiring them and the shape of the surface being scanned.

V. DISCUSSION AND CONCLUSIONS

Despite recent developments in 3D US image reconstruction algorithms and their GPU implementations, there is still a need to develop faster reconstruction methods because 3D US images must be obtained in real time in clinical applications. Hence, several approaches have been proposed to achieve better performance, and in this work, we propose an approach based on neighbor search. Our approach accelerates the reconstruction process and achieves faster image reconstruction times by optimizing the local search for information necessary to fill the gaps in 3D images during the reconstruction process. This approach uses the leaves in an octree to represent the voxels in a 3D image and uses Morton keys to efficiently track the voxel positions in the data structure. This approach enables more direct implementations of reconstruction algorithms. Notably, the methods implemented with our proposed approach achieve faster reconstruction times and use memory more efficiently than previous methods. Our analysis of our two implementations suggests that the pixel-based method performs better because it depends more on the voxels in the reconstructed 3D image. This information allows our method to take full advantage of the octree representation with Morton keys, resulting in a more direct and efficient implementation that takes utilizes the parallel processing capabilities of GPUs.

Errors may occur in reconstructed 3D US images due to various factors, such as the interpolation method used to obtain the voxel's final value or the size of the kernel used for the neighbor search. Our results indicate that we can produce complete reconstructed 3D images even when 60% of the information is missing, although the images contain some artifacts. The shape, form, and number of these artifacts depend on the number and size of the gaps in the reconstructed 3D image and the size of the kernel. In our experiments, we can observe some blurred borders, but the general appearance of the 3D images is preserved. The blurred borders in the 2D images during the reconstruction process are considered artifacts, and these blurred borders are why we report a reconstruction error greater than zero when no information is removed from the ground-truth 2D image datasets. Between our two methods, the VVN-OM method obtains worse reconstruction accuracy even when no information is removed; however, this method produces images that are more visually appealing in some circumstances (see Figure 8(d)). This visual effect occurs because of how the VVN-OM method works; the method averages the voxel values within the kernel to assign the final values in the 3D image. In our methodology, this process modifies the values in the ground-truth 3D image, leading to differences between the ground-truth image and its corresponding reconstructed 3D image. Thus, a larger error is found when the two images are compared. Our experiments show that parallelization and the use of octrees and Morton keys do not lead to significant errors in comparison with the parallel versions without octrees.

While octrees offer several advantages, an important limitation is that at a given level, the represented space is equally divided among the child nodes. When the distribution of points is unbalanced, it might be necessary to increase the octree's depth to obtain the necessary resolution. This clearly increases the processing time because more voxels are processed in each direction. Nonetheless, based on our results, the longest time was 79.79 s for the PVN-OM method to produce a 3D image with 512 × 512×512 voxels based on an input set of 131 2D US images with dimensions of 387 × 400 px. On the other hand, the same method needed 0.44 s to produce a 3D image with dimensions of 128 × 128×128 voxels based on an input set of 51 images with dimensions of 387 × 400 px, corresponding to a frame rate of 2 fps. This rate is acceptable for performing reconstruction while scanning US images. Importantly, this is an acceptable time for preoperative planning and even for most surgical procedures. Clinically acceptable reconstruction times are on the order of seconds because these values do not significantly

extend the duration of 3D US imaging studies. For instance, Hiep et al. [24] reported that 2D US image acquisition times from 3.5 ± 0.8 to 16.9 ± 6.9 minutes are considered acceptable overhead times for surgical procedures. On the other hand, Atesko et al. [23] reported that for intraoperative procedures, 14 min is the average time added due to assisted 2D navigation; this time does not represent a significant burden for surgeons, although it is not insignificant. Finally, the review presented by Zheng and Li [25] reported times for preoperative preparations (including acquisition and data processing) for several surgical procedures, and most of them take hours. This evidence indicates that there is no established time for every surgical procedure, and our reported times are within the constraints of these procedures. Furthermore, our methods have a relatively low memory consumption (although their memory requirements increase with the depth of the octree), with a maximum memory usage of 3.554 GiB. This memory requirement can be easily met with recent and unexceptional commercially available GPUs. Hence, the proposed approach is a viable option for obtaining 3D US images in real time based on a set of 2D FUS images.

While our objective is to improve the speed of 3D image reconstruction methods using neighbor search, we compared the reconstruction performance of our methods with the performance of a method using Bézier interpolation for the same reconstruction task. Our implementation of the Bézier-based algorithm produces 3D images faster than our two proposed methods, but our experiments suggest that our pixel-based method produces 3D images with fewer errors. Additionally, during our experiments, we discovered that the Bézier-based method is more sensitive to the orientation of the acquired 2D US images, while our two proposed methods are more robust with respect to the image orientation. This situation is relevant in clinical environments, where restricting the direction of scanning might not be practical. Hence, our approach is a good alternative to Bézier-based methods and a competitive choice for reconstructing 3D FUS images.

Our methods were implemented using Nvidia® hardware and CUDA®, commercial multithreading off-the-shelf GPUs and their APIs. This hardware allows voxels and points to be processed with a single CUDA® thread (depending on the kernel). In our implementations, the final reconstruction task is carried out by CUDA® kernels executing a single thread for every node in the octree to be processed. Every thread searches for points in neighboring nodes and processes the necessary data, and every thread (i.e., octree node processing) is executed in parallel by the GPU core. This data independence enables a scalable design that depends on the number of available GPU cores and their processing characteristics. For example, the RTX 2060 GPU used in some of our experiments has a total of 1,920 CUDA® cores; thus, our methods can use an octree with 2,097,152 nodes ($128 \times 128 \times 128$ voxels) requiring 1,093 "calls" to finalize the reconstruction results, whereas the GTX 1080 GPU has 2,560 CUDA® cores, which would require 820 "calls" to reconstruct the same octree.

Our experiments show that our approach accelerates the reconstruction of 3D US images and can improve the reconstruction times of pixel- and voxel-based neighbor search methods. However, as previously mentioned, the acceleration differs for the two methods, with the pixel-based method outperforming the voxel-based method. Importantly, our methods using octrees with Morton keys can produce 3D US images in real time without reducing image quality. Notably, our methods are scalable to meet changes in hardware, and they do not require large amounts of memory. Furthermore, they can obtain larger FOVs than direct 3D US systems used in clinical settings, making them an affordable alternative to direct 3D US systems. Additionally, our reconstruction time results suggest that our neighbor search-based methods have similar time performance as the Bézier-based method, although our methods are slightly slower. Thus, we can conclude that our methods are applicable in various image-assisted surgical procedures (including clinical systems and applications requiring the reconstruction and visualization of 3D US images) and are suitable for use in clinical settings. The additional time required to obtain a 3D US image with our methods does not represent a significant overhead for surgical procedures, during which medical experts have limited time to examine patients and it is frequently necessary to repeat scans to assess injuries and diseases. Additionally, we suggest that some of the techniques presented here can be incorporated into Bézier-based methods, which will be explored in future work.

ACKNOWLEDGMENT

The authors would like to thank Guillaume Zahnd for helpful advice regarding this manuscript.

REFERENCES

[1] J. R. A. Jensen, "Medical ultrasound imaging," *Prog. Biophys. Mol. Biol.*, vol. 93, pp. 153–165, Oct. 2007.
[2] M. Halliwell, "A tutorial on ultrasonic physics and imaging techniques," *Proc. Inst. Mech. Eng., H, J. Eng. Med.*, vol. 224, no. 2, pp. 127–142, Nov. 2009.
[3] A. Fenster, G. Parraga, and J. Bax, "Three-dimensional ultrasound scanning," *Interface Focus*, vol. 1, no. 4, pp. 503–519, Jun. 2011.
[4] D. H. Turnbull and F. S. Foster, "Fabrication and characterization of transducer elements in two-dimensional arrays for medical ultrasound imaging," *IEEE Trans. Ultrason., Ferroelectr, Freq. Control*, vol. 39, no. 4, pp. 464–475, Jul. 1992.
[5] Q. Huang and Z. Zeng, "A review on real-time 3D ultrasound imaging technology," *BioMed Res. Int.*, vol. 2017, pp. 1–20, Mar. 2017.
[6] C. J. Cheung, G.-Q. Zhou, S.-Y. Law, T.-M. Mak, K.-L. Lai, and Y.-P. Zheng, "Ultrasound volume projection imaging for assessment of scoliosis," *IEEE Trans. Med. Imag.*, vol. 34, no. 8, pp. 1760–1768, Aug. 2015.
[7] D. G. Gobbi and T. M. Peters, "Interactive intra-operative 3D ultrasound reconstruction and visualization," in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2002*, T. Dohi and R. Kikinis, Eds. Berlin, Germany: Springer, 2002, pp. 156–163.
[8] H. Neshat, D. W. Cool, K. Barker, L. Gardi, N. Kakani, and A. Fenster, "A 3D ultrasound scanning system for image guided liver interventions," *Med. Phys.*, vol. 40, no. 11, Oct. 2013, Art. no. 112903.
[9] P. Coupé, P. Hellier, N. Azzabou, and C. Barillot, "3D freehand ultrasound reconstruction based on probe trajectory," in *Medical Image Computing and Computer-Assisted Intervention MICCAI*. Berlin, Germany: Springer, 2005, pp. 597–604.

[10] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU—Past, present and future," *Med. Image Anal.*, vol. 17, no. 8, pp. 1073–1094, Dec. 2013.
[11] Y. Dai, J. Tian, D. Dong, G. Yan, and H. Zheng, "Real-time visualized freehand 3D ultrasound reconstruction based on GPU," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 6, pp. 1338–1345, Nov. 2010.
[12] H. Moon, G. Ju, S. Park, and H. Shin, "3D freehand ultrasound reconstruction using a piecewise smooth Markov random field," *Comput. Vis. Image Understand.*, vol. 151, pp. 101–113, Oct. 2016.
[13] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," IBM Ltd., Ottawa, ON, Canada, Tech. Rep., Mar. 1966. [Online]. Available: https://dominoweb.draco.res.ibm.com/0dabf9473b9c86d48525779800566a39.html
[14] J. Behley, V. Steinhage, and A. B. Cremers, "Efficient radius neighbor search in three-dimensional point clouds," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 3625–3630.
[15] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, MA, USA: Addison-Wesley, 1990.
[16] Q. Huang, Y. Huang, W. Hu, and X. Li, "Bezier interpolation for 3-D freehand ultrasound," *IEEE Trans. Human-Mach. Syst.*, vol. 45, no. 3, pp. 385–392, Jun. 2015.
[17] O. V. Solberg, F. Lindseth, H. Torp, R. E. Blake, and T. A. N. Hernes, "Freehand 3D ultrasound reconstruction algorithms—A review," *Ultrasound Med. Biol.*, vol. 33, no. 7, pp. 991–1009, Jul. 2007.
[18] J. W. Trobaugh, D. J. Trobaugh, and W. D. Richard, "Three-dimensional imaging with stereotactic ultrasonography," *Computerized Med. Imag. Graph.*, vol. 18, no. 5, pp. 315–323, Sep. 1994.
[19] W. Song, S. Hua, Z. Ou, H. An, and K. Song, "Octree based representation and volume rendering of three-dimensional medical data sets," in *Proc. Int. Conf. Biomed. Eng. Informat.*, vol. 1, May 2008, pp. 316–320.
[20] S. Laine and T. Karras, "Efficient sparse voxel octrees," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 8, pp. 1048–1059, Aug. 2011.
[21] R. W. Prager, R. N. Rohling, A. H. Gee, and L. Berman, "Rapid calibration for 3-D freehand ultrasound," *Ultrasound Med. Biol.*, vol. 24, no. 6, pp. 855–869, Jul. 1998.
[22] A. Rodtook, K. Kirimasthong, W. Lohitvisate, and S. S. Makhanov, "Automatic initialization of active contours and level set method in ultrasound images of breast abnormalities," *Pattern Recognit.*, vol. 79, pp. 172–182, Jul. 2018.
[23] K. Atesok, J. Finkelstein, A. Khoury, M. Liebergall, and R. Mosheiff, "CT (ISO-C-3D) image based computer assisted navigation in trauma surgery: A preliminary report," *Injury Extra*, vol. 39, no. 2, pp. 39–43, Feb. 2008.
[24] M. A. J. Hiep, W. J. Heerink, H. C. Groen, and T. J. C. Ruers, "Feasibility of tracked ultrasound registration for pelvic–abdominal tumor navigation: A patient study," *Int. J. Comput. Assist. Radiol. Surg.*, 2023, doi: 10.1007/s11548-023-02937-8.
[25] G. Zheng and S. Li, Eds., *Computational Radiology for Orthopaedic Interventions*. Cham, Switzerland: Springer, 2016.

CÉSAR VICTORIA was born in Oaxaca, Mexico, in 1990. He received the B.S. degree in computer engineering from Universidad Tecnológica de la Mixteca, Mexico, in 2012, and the M.S. degree in computer science from Universidad Nacional Autónoma de México, México, in 2016, where he is currently pursuing the Ph.D. degree in computer graphics and image processing with Posgrado de Ciencia e Ingeniería de la Computación.

From 2015 to 2017, he was a Research Assistant with the University of Auckland, New Zealand. His research interests include the development of methods for visualizing large amounts of data, focusing on free mesh algorithms, medical simulators for surgical training, and 3D reconstruction from different medical imaging modalities.

FABIAN TORRES is currently a Postdoctoral Researcher with the Physics Institute, UNAM, Mexico. His research interests include artificial intelligence applications for the segmentation, classification, and reconstruction of medical images of different modalities, such as ultrasound, computed tomography, and magnetic resonance imaging.

EDGAR GARDUÑO (Member, IEEE) received the B.Sc. degree in computer engineering from Universidad Nacional Autónoma de México (UNAM), Mexico, in 1995, and the M.S. and Ph.D. degrees in bioengineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1998 and 2002, respectively.

From 1996 to 2000, he was with the Medical Image Processing Group, University of Pennsylvania. From 2001 to 2016, he was a member of the Discrete Imaging and Graphics Group, The City University of New York, New York, NY, USA. He is currently an Assistant Professor with the Department of Computer Science, Applied Mathematical and Systems Research Institute, and with the Graduate School in Computer Science and Engineering, UNAM. His research interests include biomedical imaging, biomedical visualization, biomedical image processing and applications, projection image reconstruction applications, computer graphics, and computer vision.

FERNANDO ARÁMBULA COSÍO received the degree from the Faculty of Engineering, Universidad Nacional Autónoma de México (UNAM), in 1991, the M.Sc. degree in electronic instrumentation from the University of Manchester, U.K., in 1992, and the Ph.D. degree in medical robotics from the Imperial College of Science, Technology and Medicine, U.K., in 1997.

He was the Head of the Postgraduate Program in Computer Science, from 2008 to 2013. He is currently a Lecturer with the Institute of Applied Mathematics and Systems, UNAM. He has been teaching and supervising thesis work as part of the Postgraduate Programs in Computer Science and Engineering, UNAM, since 1999. He has published 24 articles published in several major journals in the field of medical image analysis and computer-assisted surgery and more than 40 papers presented at related international conferences. He is a coauthor of an international patent for an automatic system for microscopic image analysis of water samples.

ALFONSO GASTELUM-STROZZI received the master's degree in medical physics and the Ph.D. degree in computer science from the University of Auckland, New Zealand, in 2012.

He completed postdoctoral work with the LTHE Laboratory, Grenoble, France, where he developed methods and algorithms for simulating flows in porous media, with applications in soil models. These developments were used as part of the International Horizon 2020 project, for which he oversaw the research group in Mexico. He is currently an Associate Researcher with the Biomedical Devices Group, ICAT-UNAM. He is a Physicist with the University of Guadalajara. He develops solutions to problems in the areas of medicine, environmental science and archaeology using methods from physics and computer science. He specializes in data processing and visualization and in the modeling and simulation of natural phenomena.