Numerically robust tetrahedron-based tomographic forward and backward projectors on parallel architectures.

Ander Biguri, Hossein Towsyfyan, Richard Boardman, Thomas Blumensath

***Abstract*—X-ray tomographic reconstruction typically uses voxel basis functions to represent volumetric images. Due to the structure in voxel basis representations, efficient ray-tracing methods exist allowing fast, GPU accelerated implementations. Tetrahedral mesh basis functions are a valuable alternative to voxel based image representations as they provide flexible, inhomogeneous partitionings which can be used to provide reconstructions with reduced numbers of elements or with arbitrarily fine object surface representations. We thus present a robust parallelizable ray-tracing method for volumetric tetrahedral domains developed specifically for Computed Tomography image reconstruction. Tomographic image reconstruction requires algorithms that are robust to numerical errors in floating point arithmetic whilst typical data sizes encountered in tomography require the algorithm to be parallelisable in GPUs which leads to additional constraints on algorithm choices. Based on these considerations, this article presents numerical solutions to the design of efficient ray-tracing algorithms for the projection and backprojection operations. Initial reconstruction results using CAD data to define a triangulation of the domain demonstrate the advantages of our method and contrast tetrahedral mesh based reconstructions to voxel based methods.**

***Index terms*—** Computed Tomography, GPU, Ray tracing

I. Introduction

Pixels (or voxels in the 3D case) are the most common basis functions used for image representation, their regular grid structure is advantageous for both hardware and software. They are used therefore by the vast majority of image-based problems, such as computed tomography (CT). However, there are alternative bases to represent a continuous scalar field. Hanson and Wecksung [1] proposed the use of local basis functions that are non-zero on a regular grid to describe the continuous domain, exploring various possibilities with particular focus on B-splines. Entezari *et al* [2] expanded the B-spline reconstruction to the Radon transform. A wider variety of radial basis functions was proposed by Lewitt [3] for iterative algorithms. These bases have been shown to perform better than voxels for image reconstruction, but are also computationally more expensive [4], as they require more parameters and generate more complex operators for optimization. Wavelets have also been proposed [5], [6], providing advantages over the voxel basis not only in terms of image quality, but also in terms of computation and storage, by using wavelet based optimization and wavelet-thresholding techniques.

In this work we explore another alternative, triangular (or tetrahedral) representations. These are heavily used in computer graphics and physical modelling (e.g. in finite element analysis). A tetrahedron-based image representation has desirable properties over regular pixels. They allow for arbitrary spatial variation in image resolution and can provide arbitrarily precise object boundaries. They can reduce the heavy memory and computational burden of high resolution scans, improve the conditioning of the inverse problem and allow for improved surface models to be derived from the data. These properties can be of interest in applications of CT to non destructing testing (NDT), which often has demanding requirements. In NDT, fast yet accurate reconstruction of an often a priori known objects is required. Tetrahedral representations here allow a reduction in memory requirements and enable the easy inclusion of a prior known sample surface information into the reconstruction process.

In order to explore the potential of tetrahedral meshes for CT reconstruction, the basic operations of any CT algorithm need to be implemented — the projection and backprojection operations. These will allow for efficient simulation and reconstruction of CT scans using the proposed image basis. However implementing computationally efficient algorithms at the scale of industrial CT comes with significant challenges. Datasets consisting of 3000 to 6000 projection images with 2000^2 pixels each are now standard.

Algorithmically speaking, tetrahedral basis meshes come with added complexity over voxel meshes. Firstly, these type of meshes are unstructured, which means that standard computational and numerical approaches typically used to accelerate CT image reconstruction [7], [8] no longer apply, as most of them exploit the regular grid structure of voxels to implement efficient numerical algorithms. Secondly robust numerical methods that are not affected by the discrete nature of floating point representations of the data are required, as the image can be represented with units that can significantly vary in size within the same mesh. It is thus important that arithmetic errors will not surpass the required precision to robustly obtain geometric parameters when computing the required path-integrals. This means that geometric limits of the triangulation need to be set. Finally, due to the data size encountered in many CT problems, computationally efficient algorithms are required that can be parallelised over multiple GPUs.

A. Biguri, H. Towsyfyan and T. Blumensath are with Institute of Sound and Vibration Research (ISVR), University of Southampton

R. Boardman is with μ-VIS X-Ray Imaging Centre, University of Southampton

We are thus interested in the development of an algorithm that is as widely applicable as possible, i.e. that puts no or few constrains on the triangulation of the domain (that is, we do not want to restrict our mesh to have tetrahedra with limited aspect ratios, or meshes with triangle density constrains). Furthermore, we want the method to work with any CT reconstruction algorithm, regardless of the mathematical methods used.

Prior research in this field is not extensive. Brankov et al. [9] and later Sitek et al. [10] proposed the use of the tetrahedral basis for Positron Emision Tomography, showing improved image quality over pixels. These results are however restricted to 2D or small scale 3D tomography, which numerically is considerably more robust. Yamanaka et al. [11] proposed a surface reconstruction model for CT using tetrahedral images as basis elements, however their method is only valid for relatively regular tetrahedral meshes of a size not much larger than the detector pixel size. The used reconstruction methods are also limited, as their approach was only valid for the FDK algorithm. Quinto's PhD thesis [12] explores triangular and tetrahedral base for images, and briefly describes a GPU algorithm in a related journal article [13]. However they use tetrahedra with limited aspect ratios and the method described in their work is thus not applicable to arbitrary triangulations and was found to be prone to failure due to numerical errors when the aspect ratios of the tetrahedra increases.

We thus propose a numerically robust, parallelisable forward and backward projector for X-ray absorption CT based on ray-tracing. Our method has no constrains on the triangulation and is applicable to arbitrary reconstruction algorithms. We provide a GPU implementation of the method using the CUDA language that can reconstruct images of industrial size at reasonable time scales. The following article will first provide more detail on the numerical problems that may arise in mesh based ray-tracing and propose robust solutions that allow efficient GPU implementations. Numerical results highlight the need for our numerical approach. Results contrasting the difference in tomographic reconstructions achieved with mesh based and voxel based methods highlight potential advantages of the mesh based representations, further motivating our novel approach.

II. Methods

In this section the relevant CT background is introduced and techniques for the calculation of tetrahedron-ray intersection are discussed. A robust numerical method for forward and backpropagation is derived, followed by a discussion of technical aspects of accelerating the method on GPUs.

A. Computed Tomography

CT reconstruction is an ill posed inverse problem that can be solved using a wide variety of algorithms. The most common approach is the so-called FDK [14] algorithm that consist of two steps: high pass filtering of the measured data and backprojection of the result. Alternatively, the problem can be posed as a linear system of equations of the form

$$\mathbf{Ax} = \mathbf{b} + \tilde{\mathbf{e}}, \tag{1}$$

where $\mathbf{A}$ is a matrix where each entry represents the length of the line-element intersection between a voxel (or tetrahedron) and one of the X ray beams, $\mathbf{x}$ is the lexicographically ordered attenuation values associated with each voxel (or tetrahedron), $\mathbf{b}$ is the lexicographically ordered measured data and $\tilde{\mathbf{e}}$ accounts for errors in the data and the linearization of the problem. This equation can be solves using a wide variety of algorithms that exist in the literature including SART [15], CGLS [16], etc. All algorithms require the computation of $\mathbf{Ax}$ (the projection operation) and $\mathbf{A}^T\mathbf{b}$ (the backprojection operations), with the exception of FDK which only requires the latter. The projection operator simulates each path of an X-ray beam that is measured by a pixel in the detector and integrate the piecewise attenuation coefficients of the image x_i using the intersection length a_{ij} as

$$\hat{b}_j = \sum_{i=1}^{n_{image}} a_{ij} x_i. \tag{2}$$

Similarly the backprojection accumulates in each element of the image x_i the value from the detector considering the intersection length as

$$\hat{x}_i = \sum_{j=1}^{n_{detector}} a_{ij} b_i. \tag{3}$$

For most realistic x-ray tomography problems, the matrix $\mathbf{A}$ is extremely large and it is therefore common not to precompute and store it, but to compute the projection and the backprojection directly, by calculating the elements of A on the fly when needed [17], [18]. This involves a large amount of simple, highly parallelisable arithmetic operations, which can be computed very efficiently using GPUs. Whilst this approach has been widely studied for voxels, tetrahedra require a carefully tuned algorithm to avoid numerical errors whilst remaining computationally efficient.

B. Structuring the mesh

Tetrahedral meshes describing a detailed geometry can be dense and the unstructured nature of common mesh representation is a challenge for tomography. In an unstructured mesh, every element has to be checked for intersection with each ray, which would increase the computational time by several orders of magnitude compared to the computation of the non-zero intersections in a voxel-based methods, which can be computed very efficiently. While the common representation of meshes is unstructured, there is some structure also in general meshes and more advanced mesh representation are available that allow us to more easily exploit this structure. In this work, a graph-based representation of the mesh is used, as shown in figure 1 (for the 2D version). The graph is constructed of individual elements that each contain $nD + 1$ nodes P and $nD + 1$ ordered neighbour indexes n. This representation of the mesh, while slightly more memory consuming, provides neighbourhood information and thus allows for much more efficient algorithms for ray-tracing.

In addition to the graph containing both, element and neighbourhood information, our method uses a list of those

elements that bound the triangulated space in order to allow us to efficiently find the first ray-tetrahedron intersection without the need for an exhaustive search (see section II-E).

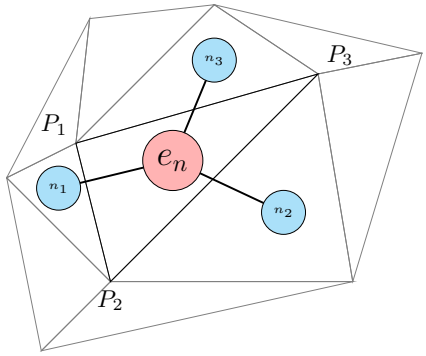


Fig. 1. Graph representation of a triangular mesh in 2D. Each element e_n contains three nodes $[P_1, P_2, P_3]$ and three neighbours $[n_1, n_2, n_3]$. The graph can be similarly built for 3D meshes.

C. Tetrahedron-Ray Intersection

A fundamental step of the algorithm is the tetrahedron-ray intersection method that computes the length of the ray within the tetrahedron, which is essentially four triangle-ray intersections as seen in figure 2. As a graph representation of the mesh is used, the index of the intersecting face of each tetrahedron can also be used to propagate the algorithm to the next intersecting element. This method however is not numerically robust when standard intersection methods are used and safeguards need to be added as discussed below.

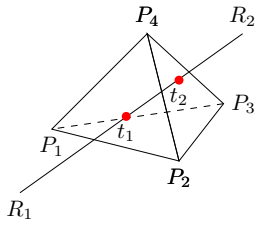


Fig. 2. The intersection algorithm computes t_1 and t_2 and the face labels denoting the faces that are intersected.

Ray-triangle intersection algorithms have been widely studied, as the computer graphics field requires fast intersections for rendering with ray-tracing. The most common method is the Möller Trumbore [19] algorithm as it is one of the fastest method available. A faster algorithm is available by Baldwin and Weber [20], however this requires precomputing and storing more data per triangle, which consumes considerably more memory in a volumetric triangulation. Both of these algorithms are based on computing the intersection location in barycentric coordinates and checking if the parameters lie inside the defined triangle. These methods however, have "leaks", as the techniques to reduce computation in the barycentric coordinate system can lead to misdirected or misrejected intersections, due to the discrete nature of the IEEE-754 [21] floating point representation of decimal numbers. Watertight algorithms have been proposed in the literature to solve this problem [22]. Watertight methods ensure that the numerical errors from the floating point arithmetic always lead to a detection as an intersection of an edge ray, which is the most desirable behaviour in computer graphics.

Both of these scenarios are undesirable in tomography. If an intersection is not detected due to leaks, the ray cannot propagate to the neighbouring element and raytracing stops prematurely. While detection of halted rays is simple, recovering from it can only be done successfully using a brute-force search of all existing tetrahedra. An example of a case where this would happen can be seen in figure 3 where a ray goes trough a node of the element. It is possible that the intersection t has not been detected due to precision errors and none of the neighbouring tetrahedra are guaranteed to detect the intersection either. The propagation algorithm would be stuck, and the only way of continuing would be to check intersections with all elements in order to find the closest intersection parameter t to the latest valid one prior to the leak. This doesn't however ensure that tetrahedra will not be missed.

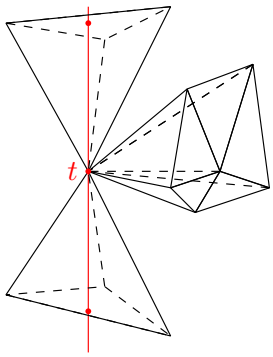


Fig. 3. Tetrahedron structure that can cause issues in the ray propagation algorithm. Assume a full convex triangulation of the domain (we here only show a selection of tetrahedra for visualization purposes). As the elements at the top and bottom are not neighbours, the algorithm propagates through zero-length intersections. If the method is not watertight, there is a risk of not finding neighbours, whilst if the method detects too many intersections (too watertight), it risks looping though neighbours infinitely.

The alternative watertight algorithm poses a different problem. Using figure 3 again as an example, a watertight algorithm would ensure that all tetrahedra touching the node are detected as intersecting with the ray. However, as all tetrahedra around the node are accepted intersections for propagation, the algorithm can get into an infinite loop. A list of intersected tetrahedra would need to be stored, but in parallel computing each processor would need to keep such a list and memory consumption would exceed realistic limitations.

The solution proposed in this article is using the Möller-Trumbore algorithm with an extra safety parameter that triggers if two intersections are not found in a tetrahedron. This safety parameter is applied in a way that effectively increases the size of the triangle faces. As the triangle increases size the intersection falls towards its interior, being safely detected as an intersection and keeping the value of the intersection parameter t unchanged. The updated method can be seen in algorithm 1, where we introduce the parameter ϵ. This method is not suitable to all ray-triangle applications, but works for

our application. If an element is checked for intersection, it guarantees that it is already known to be intersecting once, therefore increasing the triangle marginally will not have the adverse effect of accidentally including false positives. It is however important to trigger the safety parameter only when less than two intersections are found and that the safety parameter ϵ is increased gradually, as otherwise the method would fundamentally become a less accurate version of the watertight algorithm.

Algorithm 1 Möller Trumbore with safety parameter ϵ. $R_{\{1,2\}}$ define the ray and $P_{\{1,2,3\}}$ the triangle.

Require: $R_1, R_2, P_1, P_2, P_3, \epsilon$
  $\vec{d} \leftarrow R_2 - R_1$
  $\vec{E_1} \leftarrow P_2 - P_1$
  $\vec{E_2} \leftarrow P_3 - P_1$
  $\vec{q} \leftarrow \vec{d} \times \vec{E_2}$
  $a \leftarrow \vec{E_1} \cdot \vec{q}$
  if $a > -10^{-8}$ **and** $a < 10^{-8}$ **then** ▷ Check if its zero
    return false
  end if
  $f \leftarrow 1/a$
  $\vec{s} \leftarrow R_1 - P_1$
  $u \leftarrow f(\vec{s} \cdot \vec{q})$
  if $u < (0 - \epsilon)$ **then**
    return false
  end if
  $\vec{r} \leftarrow \vec{s} \times \vec{E_1}$
  $v \leftarrow f(\vec{d} \times \vec{r})$
  if $v < (0 - \epsilon)$ **or** $(u + v) > (1 + \epsilon)$ **then**
    return false
  end if
  $t \leftarrow f(\vec{E_2} \cdot \vec{r})$
  return {**true**, t}

1) Floating Point Precision: GPUs do not handle double precision floating point arithmetic fast, even those designed specifically for double precision operations remain twice as slow. Memory also plays a role. Not only does double precision require twice the memory a more important bottleneck in GPU computing tends to be memory reading and waiting times. Therefore, most arithmetic on GPUs tends to use single precision operations. This is not true for the presented algorithm. Due to the scale differences involved in tomography, it is imporatnat that the intersection code has higher precision than the data that it uses. In IEEE-754 single floating point representation, the rounding error can be roughly estimated to be in the 8th or 9th digit after the most significant decimal point. However, the order of magnitude of $\vec{d}$ and $\vec{E}_{\{1,2\}}$ in algorithm 1 can differ by the same amount. Even when the magnitudes are closer, the arithmetic operations can result in errors big enough that intersections can be mislabelled, specially when the triangles have high aspect ratios. An experiment showcasing this issue is presented in section III-C.

D. Numerically Robust Ray-Propagation

The projection and backprojection algorithms are essentially the same, with the only difference that the projection operator integrates over the path into a detector pixel, and the backprojection operator gathers the detector pixel values into the element attenuation coefficients. Algorithm 2 describes the ray-tracing method. The algorithm essentially computes the current elements' intersection length, updates the integral (or the element, in the case of the backprojection) and propagates to the element neighbouring the last intersection (t_2) plane. When the tetrahedon-ray intersection is checked there is a safety check to ensure that two faces are intersected and if not, then the safety parameter ϵ is increased, until two intersections are found. While this is a very rare event, the amount of tetrahedra and rays that tomography requires makes it statistically likely to happen. In some of our experiments this happened as rarely as five times every million rays, however due to the ill-posed nature of X-ray reconstruction, this event has noticeable negative effects on reconstruction. An extra check thus needs to be performed when zero-length intersections are found, to ensure there is no backtracking by choosing the wrong face for the propagation of the ray, which can happen when a node exist with several tetrahedra (see Fig.3).

Our algorithm has only one constraint: the volumetric mesh must be convex, however, as non-convex meshes can be convexified by the introduction of additional triangles, this is not very restrictive.

Algorithm 2 Robust ray-tracing for tetrahedon X-ray projection

Require: Geometry information, graph
  Launch Kernel for every pixel $p[i, j]$ in the detector
  $l \leftarrow \|R2 - R1\|$
  $\epsilon \leftarrow 10^{-9}, \quad \sum \leftarrow 0$
  Read initial intersection $element$ index, i_{now}
  return if $i_{now} = -1$
  while $i_{now} \neq -1$ **do**
    while not Intersection **do**
      $t_1, t_2 \leftarrow$ TetraRayIntersection(i_{now}, ϵ)
      $\epsilon \leftarrow \epsilon \cdot 10$
    end while
    $\epsilon \leftarrow 10^{-9}$
    $\sum \leftarrow \sum + l \cdot (t_2 - t_1) \cdot element[i_{now}]$
    if $t_2 = t_1$ check if they need to be swapped
    $i_{now} \leftarrow$ neighbour of face where t_2 happened
  end while
  $p[i, j] \leftarrow \sum$

E. Initialization of the Propagation Algorithm

The previous section ignores an important issue: finding the first intersecting tetrahedron to initialzie the ray-propagating algorithm. To limit the memory usage and simplify the propagation code, Algorithm 2 requires the index of a tetrahedron on the mesh boundary.

The issue with initializing arbitrarily shaped convex tetrahedra meshes is that a brute force approach is not possible. Even for optimised meshes, with large boundary elements, the number of tetrahedra on the boundary of the mesh is typically

larger than the number of times the X-ray path intersects with internal tetrahedra so that the initialization could take significantly longer than the X-ray propagation. On larger meshes, brute force initialization would thus dominate computation time. The only viable solution is to avoid checking most of the boundary elements by implementing an efficient search strategy.

Quinto el. al. [13] propose a quadtree representation of the boundary surface triangles to initialise rays propagation. This approach works very efficiently, but adds a significant topological constraint to the input mesh: it requires that surface elements are arranged on a regular mesh, with edges aligned to the quadtree axes. This implies that the shape and size of the elements are strongly constrained. As the proposed method in this article aims for a generic implementation with minimal constraints on the input mesh shape, an alternative modified approach is proposed.

In this work, an R*-tree [23] is precomputed for the boundary elements in a pre-processing step in order to accelerate the initialization procedure. An R*-tree is a depth-balanced search three that is constructed by bounding regions containing each nodes children. They are a variation of R-trees, search trees designed to contain volumetric objects, optimised for spacial access, i.e. to pack objects that are closer together by some metric. The R*-tree is a variation to the standard R-tree [24] which minimises not only the area of each node, but also the overlap between nodes. These search trees are particularly interesting for the initialization of X-rays as the depth balance of R*-trees guarantees similar computational costs for all X-ray paths, which is beneficial in parallel implementations. Additionally, the box-shaped bounding regions that define each node allow for very fast box-ray intersection checks.

To search the tree, a depth-first algorithm is implemented, as it requires minimal memory allocation per search, which can be a critical factor for GPU implementation. When a leaf node is reached in the search, all tetrahedra within that node are checked for intersection and the index of whichever has the minimum intersection parameter, i.e. the earliest intersection along the X-ray path, is stored.

R*-trees have a minimum and maximum number of children in each node, which implicitly defines the depth for a given input database. We have chosen 10 as maximum number elements and 4 as minimum, as it generates trees that are not too deep, but containing a small number of tetrahedra in each leaf node, which minimizes the number of tetrahedra that need to be checked for intersection by the depth-first method.

F. GPU Implementation Details

In GPUs, paralellization happens in small blocks of execution threads. Each thread inside a blocks is executed in parallel (not necessarily concurrently) and the GPU waits until the entire block is done before allocating a new one to the processor cores responsible for the execution. Thus, ensuring that all threads within a block execute similar code and terminate at the same time increases overall computation speed, as it minimises idle threads.

In tomography, the highest cost comes from memory reads, as they can be two orders of magnitude slower than an arithmetic operation and lead to idle threads waiting for read operations to be completed. Therefore, most of the optimizations that happen in GPU code relates to executing code that would access the same memory at the same time, to increase cache hits. Tetrahedron based tomography however is not very sensitive to this optimization, as the unstructured nature of the mesh means that even adjacent rays are not necessarily reading the same location in memory, i.e. they are not crossing the same tetrahedra nor the same amount of tetrahedra. On top of that, the nature of the algorithms proposed here means that they require several branches per execution block, so thread divergence is high (an undesired behaviour on GPU parallelization). It is likely that thread divergence hides memory latency problems as the fastest speed can be achieved by computing the rays in small square blocks. We found empirically that 8×8 blocks works best on GTX 10XX GPU models. We suggest empirically testing different size if this code is required to execute on alternative GPUs.

The backprojection is more problematic. When launching the algorithm in parallel, multiple threads want to update the same attenuation coefficients. Atomic operations ensure that no race conditions are met, but in theory this increases the time significantly as most of the threads could be waiting to write. While separating the rays may seem the best approach, empirical tests show that running the backprojection similarly to the projection gives the best results. This unintuitive behaviour is likely caused by the thread divergence as it can hide atomic writing latency. On top of that, the general nature of the algorithm and tetrahedra meshes can result in a limited amount of simultaneous write attempts, depending on the particular input mesh.

For our multi GPU approach, projections are divided, while keeping the full mesh in each of the GPUs memories. This is because dividing the graph in pieces of similar computational cost that are also convex is a significant challenge, and its size would generally be small enough to fit entirely in compute oriented GPUs.

MATLAB and CUDA code that implement our approach can be freely accessed at github.com/AnderBiguri/TriangleCT.

III. Results

To evaluate the quality and performance of the algorithm initial experiments are presented here.

A. Image Reconstruction

To test the tetrahedral mesh projector and backprojector a simulated experiment is presented. Using a tetrahedral mesh, X-ray projections are simulated and reconstructed. We here use our new tetrahedral approach and contrast it to standard voxel based reconstruction using the TIGRE toolbox [17]. The surface representation of the data used is shown in figure 4. The mesh is generated using three surfaces: a cube bounding the volume area, a closed surface version of the Stanford bunny and the Utah teapot. The models are meshed using the Simpleware ScanIP, which generates a wide rate of tetrahedral sizes and provides individual tetrahedra with high aspect ratios. The resulting mesh contains approximately

175,000 tetrahedra. Attenuation coefficients of values 0, 1 and 2 are assigned to the tetrahedra in the box, the bunny and the teapot respectively.

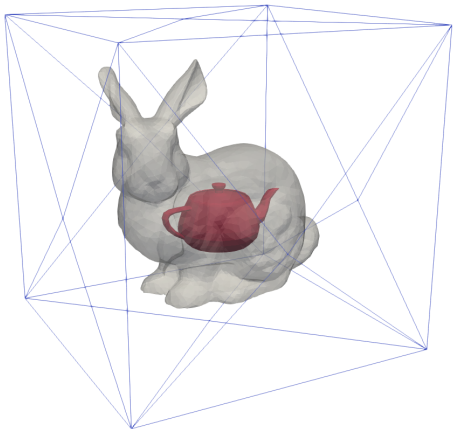


Fig. 4. Surface representation of the tetrahedra mesh used for validation of the forward and backward projector.

Using the tetrahedra based forward projector, 100 projections of 1024×1024 pixels are simulated around a circular trajectory in equidistant angles for a cone beam. Figure 5 shows projection at angle $0°$ and $90°$.

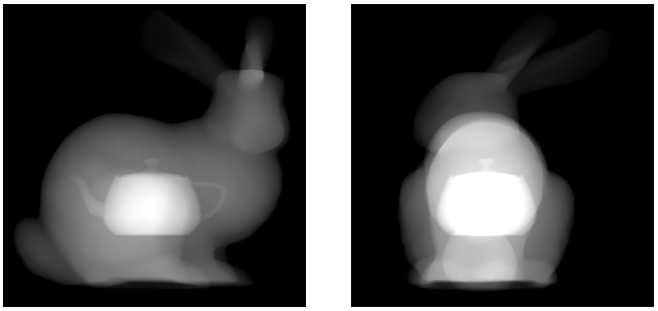
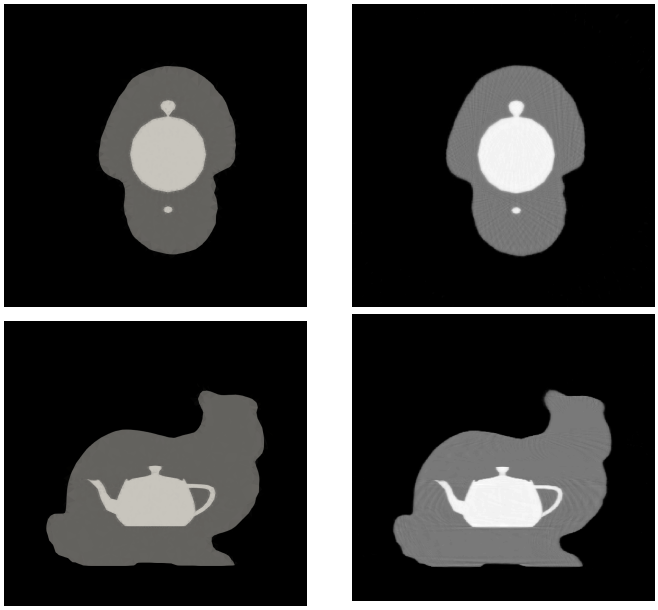


Fig. 5. Simulated X-ray projections for $0°$ and $90°$ rotation positions.

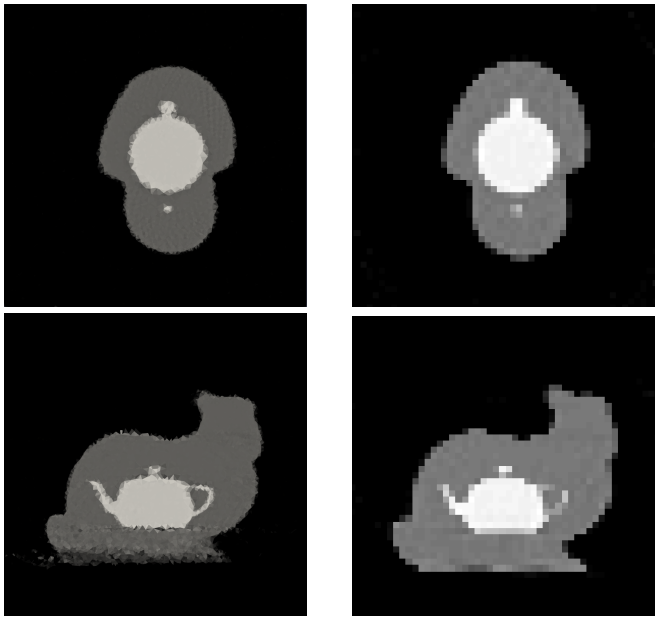
Four reconstructions are computed. On a tetrahedra basis reconstructions, the same mesh as the one used to generate the data is used, as well as a "high resolution" 3500k element mesh. For the voxel based representation a $512 \times 512 \times 512$ mesh, and a $56 \times 56 \times 56$ mesh (the same amount of elements as the original tetrahedra mesh) spanning the same volumetric area is used. OS-SART [25] is here used for reconstruction. The algorithm is run with the same parameters for both reconstruction volumes, using 50 iterations with blocks of 20 projections. Cross section results of the reconstruction are shown in figure 6.

Comparing reconstructions is non-trivial, as any comparison would necessarily need to map from one basis to the other, likely obfuscating benefits or errors of each. For example, converting the voxel basis to tetrahedral basis for comparison may hide the blur on boundaries or the visible streak artefacts on the reconstruction. On the other hand, converting from the tetrahedral basis to voxel may show that the mesh-based model reconstructs images that are more uniform, but this is just a consequence of a particularly large tetrahedra on a given experiment, not necessarily due to general robustness. However, one can observe that the nature of the errors are significantly different between the two image types. The voxel basis image shows typical streak artefacts for low angle scans or horizontal flat surfaces and a general blur, especially around small features. The tetrahedral basis image however does not



(a) Tetrahedra basis on known mesh

(b) Voxel basis in high resolution



(c) Tetrahedra basis on unknown mesh

(d) Voxel basis in low resolution

Fig. 6. Iterative reconstruction using the OS-SART algorithm with 50 iterations, block size of 20 projections with 100 projections in total. Reconstruction is shown in (a) tetrahedra basis with a known mesh (170k tetrahedra) and (b) voxels (512^3 voxels), (c) tetrahedra basis on unknown mesh (3500k tetrahedra) and (d) voxel basis on low resolution (56^3 voxels, the same amount as tetrahedra are in (a)). The attenuation coefficients are shown in range [0-2.1]

seem to be affected by the streak artefacts when the surface mesh is known. It does however, accumulate most of the errors around the boundaries of objects.

Note that we here find the best reconstruction using a tetrahedral basis using the same mesh to generate the projection data and compute the tetrahedral reconstruction. Whilst this is not realistic in real applications, it is done here to 1) show that our method does accurately compute forward and backward projections and thus provides low error reconstructions in the ideal case and 2) to demonstrate that tetrahedral meshes can have advantages over voxel basis if a good mesh is chosen, as the voxel basis has more unknown elements than observations leading to typical artefacts, whilst the mesh representation has fewer unknowns than measurements and thus does not suffer from the same problem if a good mesh is available. Obviously finding a good mesh is in itself an important issue, as can be seen in Figure 6 (c), where a random mesh seems to enhance the streak artefacts visible in the voxel meshes.

B. Image Reconstruction of a CAD model

In tetrahedral mesh tomography, the structure and shape of the tetrahedra can have mayor importance on the quality of the reconstruction. Approaches to refine the a mesh have been published (e.g. by Yamanaka et al. [11] for tetrahedra reconstruction with FDK) whilst solutions to align available CAD models to projection measured data also exist [26]. The following experiment highlights the difference between having accurate knowledge of a prior model and having an ideally aligned mesh generated by a prior model. We use the CAD model in Figure 7 to simulate 100 projections of size 1024×1024. The same model has been used to generate a tetrahedra mesh with approximately 620k elements, while a tetrahedra mesh with 980k elements is also created of the same dimensions, but without knowledge of the object boundaries.

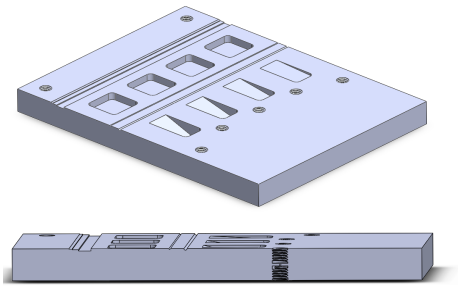
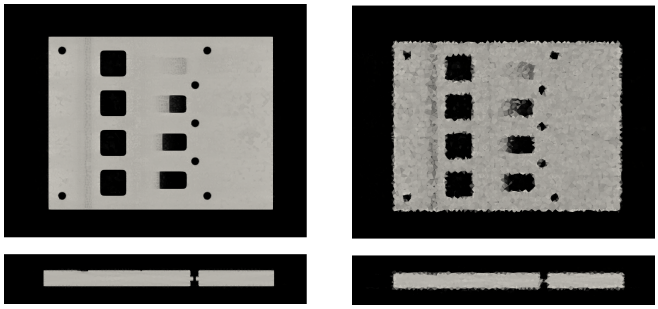


Fig. 7. CAD model designed for CT image quality and feature detection evaluation.

Figure 8 shows the reconstructionusing the two different meshes. It can be clearly seen that having prior information on the tetrahedral mesh can have major effects on reconstruction quality, even with meshes that are significantly smaller in terms of available elements.

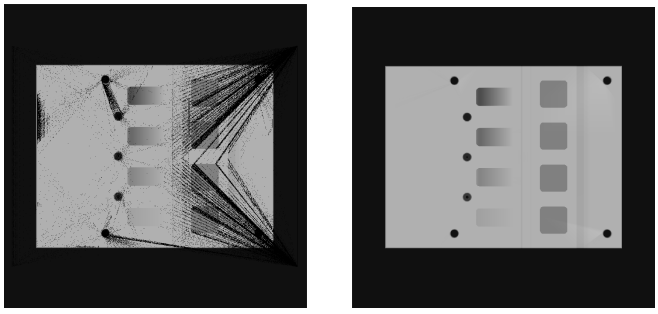
C. Floating point errors

As mentioned in section II-C1, using double precision floating point arithmetic (or, in general, higher precision numerics than the precision of the mesh) for the triangle intersection code is critical, especially in cases where the aspect ratio of the triangles is high or the triangles are very small. Figure 9 shows a projection of a single material piece that has been triangulated with a Delaunay triangulation algorithm using the points of an STL file with added nodes on the edges to ensure surface preservation. This generates a high definition triangulation with the minimal number of tetrahedra, but with a low quality mesh[1] due to high aspect ratios. This type of model can be useful to simulate projections from known data e.g. from CAD models. Results in figure 9(a) demonstrates how using the single precision numerical intersection code results in a high number of pixels where the ray-propagation integrals fail to terminate properly, either by sudden termination of the propagation (due to missed intersections) or due to infinite looping through triangle neighbourhoods.



(a) Mesh with prior information (b) Dense random mesh

Fig. 8. Iterative reconstruction using the OS-SART algorithm with 50 iterations, block size of 20 projections with 100 projections for tetrahedra mesh based images, (a) with prior surface information and (b) without prior information. The attenuation coefficients are shown in range [0-1.1]



(a) Single precision (b) Double precision

Fig. 9. Effect of floating point types in the ray-triangle intersection algorithm for a specific projection. Black dots in (a) are rays that force the algorithm to terminate abruptly or loop infinitely.

D. Performance figures

Evaluating performance is not straightforward. The forward and backprojection algorithms are effectively the same, but the sub-algorithm used for initialization is fundamentally different from the ray propagation algorithm with both performing

[1] In terms of FEM mesh quality metrics

significantly difference. As arbitrary meshes can vary significantly in the number of boundary elements compared to the number of total elements, the overal algorithm performance can either be dominated by the initialisation (if there are relatively many boundary elements) or by the ray tracing (if there are relatively few boundary elements). Even if two meshes have the same number of boundary and internal elements, the shape of these elements can cause major differences in performance of each of the two sub-algorithms, making general performance figures hard to derive.

To provide some intuition into the performance, a particular type of regular mesh is thus here used as a benchmark. We generate voxel-type regular meshes that are then triangulated. This type of mesh does not exploit the capabilities of tetrahedra mesh reconstruction but allows for reasonable performance computation and compassion. Whilst these results are not directly usable to predict performance of arbitrary meshes, an approximate estimation can be obtained by adding the results for initialization and ray-propagation for particular meshes after acounting for the n umber of boundary elements and total elements. For example, the mesh on Figure 8(a) has 6×10^5 total elements, but contained only 192 boundary elements.

Figure 10 shows computing times on a single GTX 1050 GPU for regular tetrahedra meshes of increased size. Both measurements should be interpreted separately. The regular meshes have been created by linearly increasing the number of points along each edge. The projection kernel linearly increases its computational cost with respect to edge length. This behaviour is expected, as the ray-propagation kernel is expected to encounter a number of elements that linearly increases with edge length on a regular grid, but the measurement highlights that the algorithm does not have a memory or compute limitation related to mesh size. The initialization kernel also behaves as expected. It showcases more jagged lines, caused by differences in the R*-tree generation for that particular shape and size. We hypothesize that a better R*-tree bulk loading [27] procedure may be able to generate a more stable profile, but have not tested this. The initialization procedure increases logarithmically with the edge length, which is also what is expected with tree-like search algorithms.

Memory transfer speeds are not show as they almost always had the same speed, reaching 20 ms of total time for small sizes while keeping that same total time on the largest meshes.

As the multi-GPU splitting method used is simple copying the entire memory to all GPUs and splitting the total projections between the available GPUs, no performance measures are required. The time per projection is the same, but now some projections are computed simultaneously. The minor overhead of multiple memory copies can be practically ignored due to its short length compared to total computation times.

It is worth noting that voxel-based tomographic reconstruction is able to compute a single projection from approximately 2×10^6 voxels (i.e. a 128^3 voxel image) onto the same 1024×1024 detector in around 120 ms including memory transfer. The voxel-basis projection thus remains an order of magnitude faster than a tetrahedral mesh based projection. This speed advantage is due to the regular structure in voxel based meshes. Nevertheless, using tetrahedral basis meshes enables the use of significantly smaller meshes to accurately represent images to the same level of detail. Thus while voxel based images are faster for the same level of elements, they are not necessarily so for the same level of detail.

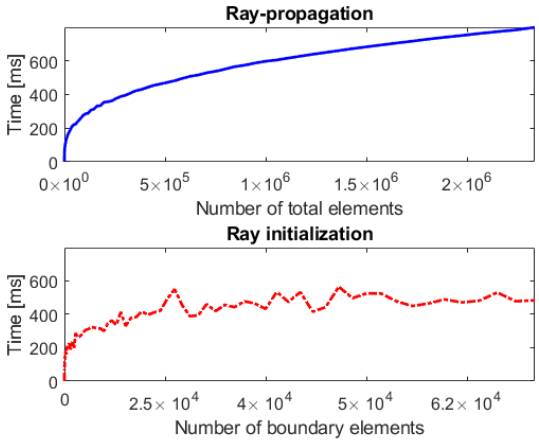


Fig. 10. Kernel initialization and ray-propagation times for a single 1024×1024 projection at different mesh sizes and boundary sizes.

IV. Discussion

We developed a numerically robust, GPUs accelerated method for X-ray forward and backprojection on convex meshes with arbitrarily shaped tetrahedral elements. Numerical results using challenging meshes shows the robustness of the new approach. We presented some comparisons to standard voxel based reconstructions, showing that better results can be obtained if good boundary mesh estimates are available. Whilst computation speed for generic meshes remains larger than that for voxel based meshes with the same number of elements, tetrahedral meshes have significantly increased flexibility so allow meshes with significantly fewer elements to be used without a decrease in image quality.

The focus here has been on the description of the numerical GPU algorithm and the demonstration of its computational capabilities. Having developed this new tool, there are now a range of interesting open questions for further work. Many concepts applied to voxels can be used with tetrahedra too, such as applying shape functions, which assume non-uniform voxel value. Multi-grid or multi-scale method can also be applied to the tetrahedral basis, as the elements can be split and refined at different scales in the same way as voxel-based grids. The application of typical image priors (such as total variation norms) to provide regularized image reconstruction is also possible. In voxel images, the image gradients are discreatized on a regular grid using simple finite differences methods, however the intrinsic irregularity of tetrahedra meshes may require a more complex gradient discretization.

Our tool allows efficient simulation of transmission tomography from mesh based representations. The method can, for example, be extended to generate accurate X-ray projections from multi-material volumes, for example using Monte Carlo photon simulations. Whilst most existing software is based on surface models, using our volumetric model would allow

for the specification of more gradual material changes with arbitrary surfaces.

Tetrahedra based image reconstruction also has significant potential in image reconstruction applications where the sample geometry is known, as we already highlighted in [28], where the mesh model seems to outperform voxel-based methods on limited angle scans thanks to the use of prior surface information. The next steps to test the mesh-based method is to use it together with algorithms that align prior CAD models to projection data and then refine a tetrahedral mesh for surface characterization within the reconstruction process. Together with iterative reconstruction capabilities, these two methods have the potential to allow reconstruction from limited numbers of projections so that fault detection can be performed fast and reliably in in-process CT applications. Similarly, the tetrahedral reconstruction could be of benefit in image-based modelling applications, where a sample is scanned in order to create a physical model to be simulated. These models are often based on finite element methods and thus require a tetrahedral basis, which are currently generated using segmentation techniques on voxel-based reconstructions. By directly reconstructing with a tetrahedral basis, more accurate surfaces estimates may be possible.

ACKNOWLEDGEMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. This research was supported by EPSRC grant EP/R002495/1 and the European Metrology Research Programme through grant 17IND08. A. Biguri would like to thank Marco Vallario and Tobias Bertel for interesting and productive discussions on ray tracing and search trees.

REFERENCES

[1] Kenneth M Hanson and George W Wecksung, "Local basis-function approach to computed tomography," *Applied Optics*, vol. 24, no. 23, pp. 4028–4039, 1985.

[2] A. Entezari, M. Nilchian, and M. Unser, "A box spline calculus for the discretization of computed tomography reconstruction problems," *IEEE Transactions on Medical Imaging*, vol. 31, no. 8, pp. 1532–1541, Aug 2012.

[3] R M Lewitt, "Alternatives to voxels for image representation in iterative reconstruction algorithms," *Physics in Medicine and Biology*, vol. 37, no. 3, pp. 705–716, mar 1992.

[4] Samuel Matej and Robert M Lewitt, "Practical considerations for 3-d image reconstruction using spherically symmetric volume elements," *IEEE Transactions on Medical Imaging*, vol. 15, no. 1, pp. 68–78, 1996.

[5] Maaria Rantala, Simopekka Vanska, Seppo Jarvenpaa, Martti Kalke, Matti Lassas, Jan Moberg, and Samuli Siltanen, "Wavelet-based reconstruction for limited-angle x-ray tomography," *IEEE transactions on medical imaging*, vol. 25, no. 2, pp. 210–217, 2006.

[6] Shiying Zhao and Ge Wang, "Feldkamp-type cone-beam tomography in the wavelet framework," *IEEE transactions on medical imaging*, vol. 19, no. 9, pp. 922–929, 2000.

[7] Cheng-Ying Chou, Yi-Yen Chuo, Yukai Hung, and Weichung Wang, "A fast forward projection using multithreads for multirays on GPUs in medical image reconstruction," *Medical Physics*, vol. 38, no. 7, pp. 4052–4065, 2011.

[8] Timo Zinsser and Benjamin Keck, "Systematic performance optimization of cone-beam back-projection on the Kepler architecture," .

[9] Jovan G Brankov, Yongyi Yang, and Miles N Wernick, "Tomographic image reconstruction based on a content-adaptive mesh model," *IEEE Transactions on medical imaging*, vol. 23, no. 2, pp. 202–212, 2004.

[10] Arkadiusz Sitek, Ronald H. Huesman, and Grant T. Gullberg, "Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud," *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1172–1179, sep 2006.

[11] Daiki Yamanaka, Yutaka Ohtake, and Hiromasa Suzuki, "The sinogram polygonizer for reconstructing 3D shapes," *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[12] Michele Arcangelo Quinto, *Méthode de reconstruction adaptive en tomographie par rayons X: optimisation sur architectures parallèles de type GPU*, Grenoble, Apr 2013.

[13] Michele Arcangelo Quinto, Dominique Houzet, and Fanny Buyens, "Tetrahedral volume reconstruction in X-ray tomography using GPU architecture," *2013 Conference on Design and Architectures for Signal and Image Processing*, pp. 334–339, 2013.

[14] L.A. Feldkamp, L.C. Davis, and J.W. Kress, "Practical cone-beam algorithm," *J. Opt. Soc. Am. A*, vol. 1, no. 6, pp. 612–619, Jun 1984.

[15] A.H. Andersen and A.C. Kak, "Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm," *Ultrasonic imaging*, vol. 6, no. 1, pp. 81–94, 1984.

[16] A. Bjorck, *Numerical Methods for Least Squares Problems*, Society for Industrial and Applied Mathematics, 1996.

[17] Ander Biguri, Steven Hancock, Manjit Dosanjh, and Manuchehr Soleimani, "TIGRE: A MATLAB-GPU toolbox for CBCT image reconstruction," *Biomedical Physics & Engineering Express*, vol. 2, no. 5, pp. 055010, 2016.

[18] Wim van Aarle, Willem Jan Palenstijn, Jeroen Cant, Eline Janssens, Folkert Bleichrodt, Andrei Dabravolski, Jan De Beenhouwer, K Joost Batenburg, and Jan Sijbers, "Fast and flexible x-ray tomography using the ASTRA toolbox," *Optics express*, vol. 24, no. 22, pp. 25129–25147, 2016.

[19] Tomas Möller and Ben Trumbore, "Fast, minimum storage ray/triangle intersection," in *ACM SIGGRAPH 2005 Courses*. ACM, 2005, p. 7.

[20] Doug Baldwin and Michael Weber, "Fast ray-triangle intersections by coordinate transformation," *Journal of Computer Graphics Techniques*, vol. 5, no. 3, 2016.

[21] *IEEE standard for binary floating-point arithmetic*, Institute of Electrical and Electronics Engineers, New York, 1985, Note: Standard 754–1985.

[22] Sven Woop, Carsten Benthin, and Ingo Wald, "Watertight ray/triangle intersection," *Journal of Computer Graphics Techniques*, vol. 2, no. 1, 2013.

[23] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *Acm Sigmod Record*. Acm, 1990, vol. 19, pp. 322–331.

[24] Antonin Guttman, *R-trees: A dynamic index structure for spatial searching*, vol. 14, ACM, 1984.

[25] Y. Censor and T. Elfving, "Block-iterative algorithms with diagonally scaled oblique projections for the linear feasibility problem," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, no. 1, pp. 40–58, 2002.

[26] Tasuku Ito, Yutaka Ohtake, Yukie Nagai, and Hiromasa Suzuki, "Thickness measurement of metal plate using CT projection images and nominal shape," in *9th Conference on Industrial Computed Tomography*. Feb. 2019, NDT.

[27] Ibrahim Kamel and Christos Faloutsos, "On packing r-trees," University of Maryland.

[28] Ander Biguri, Hossein Towsyfyan, Richard Broadman, and Thomas Blumensath, "Initial result on the use of tetrahedra-based imaging for limited angle tomography," in *9th Conference on Industrial Computed Tomography*. Feb. 2019, NDT.