Tetrahedral Volume Reconstruction in X-Ray Tomography using GPU Architecture

Michele Quinto, Dominique Houzet, Fanny Buyens

Tetrahedral Volume Reconstruction in X-Ray Tomography using GPU architecture

Michele A. Quinto
CEA, LIST,
91191 Gif-sur-Yvette, France

Dominique Houzet
GIPSA-lab, Grenoble-INP,
38402 St. Martin d'Heres, France
Email:dominique.houzet@gipsa-lab.grenoble-inp.fr

Fanny Buyens
CEA, LIST,
91191 Gif-sur-Yvette, France
Email: fanny.buyens@cea.fr

***Abstract*—In this paper, we propose the use of the graphics processor unit (GPU) to accelerate a ray-tracing method in the framework of X-ray tomographic image reconstruction. We first describe an innovative iterative reconstruction method we have developed based on a tetrahedral volume with conjugate gradient. We do not use voxels here but instead tetrahedrons to increase the quality of reconstruction and the reduction of data as thus we need less resolution of the volume to fit the object reconstructed. This is an important point to use the GPU. We present here the algorithms adapted to the GPU and the results obtained compared to CPU.**

I. Introduction

Tomography reconstruction from projections data is an inverse problem widely used in the medical imaging field. With sufficiently large number of projections over the required angle, the FBP (Filtered Back-Projection) algorithms allow fast and accurate reconstructions. However in the cases of limited views (low dose imaging) and/or limited angle (specific constrains of the setup), the data available for inversion are not complete, the problem becomes more ill-conditioned and the results show significant artifacts. In these situations, an alternative approach of reconstruction, based on a discrete model of the problem, consists in using an iterative algorithm or a statistical modelisation of the problem to compute an estimate of the unknown object. These methods are classicaly based on a volume discretization into a set of voxels and provide 3D maps of densities. High computation time and memory storage are their main disadvantages. Moreover, whatever the application is, the volumes are segmented for a quantitative analysis. Numerous methods of segmentation with different interpretations of the contours and various minimized energy functions are offered with results that can depend on their use by the application.

This work presents a novel approach of tomographic reconstruction simultaneously to segmentation of the different materials of the object to reconstruct. The process of reconstruction is no more based on a regular grid of voxels but on a mesh composed of non regular tetraedra. After an initialization step, the method runs into three main steps: reconstruction, segmentation and adaptation of the mesh, that iteratively alternate until convergence. For that purpose, we have adapted and optimized two iterative algorithms of reconstruction (OSEM and Conjgate Gradient) usually used in a conventionnal way to be performed on irregular grids of triangular or tetraedric elements. For the segmentation step, a geometrical approach (level set) was implemented. The adaptation of the mesh to the content of the estimated image is then constrained by the contours segmented the step before that makes it progressively coarse from the edges of the object to the limits of the domain of reconstruction. At the end of the process, the result is a classical tomographic image in gray levels, whose representation by an adaptive mesh to its content provides a corresponding segmentation. The results show that the method provides reliable reconstructions and leads to drastically decrease the memory storage. In this context, both the 3D operators of projection and backprojection were implemented on parallel architectures namely the graphic processor units (GPU). These operators are based on the ray-tracing method used in image rendering.

II. X-Ray Tomography

X-rays tomography is a non invasive technique which provides 3D reconstruction of an object. Its principle is based on the interaction of the X-ray beams with the materials composing the object, and the counting of the transmitted photons by detectors placed in front of the X-ray tube. The acquired data are collected according to multiple orientations (angles) of the source-detector system over the object. The number of projections and the rotation step vary according to the application and the type of devices. By means of these data, an image in grey levels is mathematically reconstructed representing the internal density of the object.

Commonly, the algebraic algorithms used for reconstruction are based on a representation of the volume by a regular lattice of voxels. However, the spatial resolution of the detectors increases and consequently the number of voxels too to preserve the high spatial resolution. The number of unknowns to estimate can thus be very important, and the algebraic methods of reconstruction very time consuming.

The algebraic methods model the inverse problem by a discrete approach: the projections are interpreted as samples of a function with discrete variables, and the object is represented as an unknown function f defined as a linear combination of basic functions (commonly characterizing voxels). So the inverse problem is written as a linear system:

$$p = H\,f \tag{1}$$

where the vector p contains the measures of the projections. The size of the vector p is equal to $N_D \times N_p$, where N_D is the number of the pixel detector and N_p the number of projections. H is the matrix system ; it depends on the geometry of acquisition. The value of its elements h_{ij} represents the contribution of element j to X-ray beam projection i. Commonly, h_{ij} is equal to 1 if pixel j is intersected by X-ray beam i, otherwise 0, but other expressions of contribution can be used as, for example, the length or the surface of the X-ray beam intercepted in the pixel. Matrix H is very sparse : most of its elements are zero. Nevertheless, the computation of matrix H is very time consuming. In the case of small data sets, H can be pre-calculated and stored in memory. Although, as the size of the data sets enlarge, matrix H cannot be stored in memory and several approaches have been proposed to make it more compact. For example, [1] proposed a list-mode approach to only store the non-zero elements of the matrix. A more efficient solution is to calculate the coefficients of matrix H on the fly using ray-tracing methods. The optimization of matrix H computation improve the performances of both the projection and the back projection which are the main operators of algebraic algorithms of reconstruction. The approaches used to compute projection and backprojection are classified as [2]:

- ray-driven: the X-ray beams are centered on the pixels of the detector
- pixel-driven: the X-ray beams are centered on the pixels of the volume of reconstruction.

In this work, the two approaches were implemented and optimized on GPU (Graphic Processor Unit, multi-threaded parallel architectures), to study the parallelisation and optimisation characteristics of both the projector and the back projector (the latest being the transposed operator of the first one). In the case of the ray-driven implementation, the projector performs gather operations by reading the data and writing the result to a specific memory address (corresponding to the thread IDs), while the backprojector performs scatter operations by updating the values of the estimated image shared by several threads. On the contrary, in the case of the pixel-driven implementation, the projector performs scatter operations by ... while the backprojector performs gather operations by ... . For more efficiency, gather operations are prefered to ensure no memory writing conflicts. In this way, we implemented a ray-driven approach for the two operators [3], the projection requiring more operations than the backprojection. The small number of detectors induced a limited amount of exploitable parallelism but a sufficient one for a single GPU.

Table I: Memory write operations.

	ray-driven	voxel-driven
Projector	Gather	Scatter
Backprojector	Scatter	Gather

III. Proposed Method

In this work, we proposed a method of tomography reconstruction combined with a stage of segmentation, that results to a classical image of reconstruction in grey levels paired with an image of segmentation of the studied object. In this method, we adopted a mesh representation based on tetrahedral elements to discretize the volume of reconstruction in order to obtain a

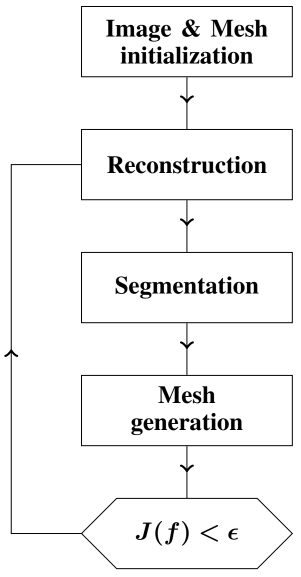


Figure 1: Diagram of the proposed method.

better representation of convex objets and to drastically reduce the number of elements during the process of reconstruction (the mesh being adapted to the content of the estimated image). Thus, as we do not consider anymore a three-dimensional volume described by voxels but a set of irregular tetrahedra, common used algebraic algorithms of reconstruction as EM and Conjugate Gradient were implemented and optimized to be used on irregular grids of tetraedra.

The proposed method, we refer as ATM (Adaptive Triangular Mesh reconstruction), is composed of three main stages after initialization : reconstruction, segmentation and adaptation of the mesh which successively alternate until convergence (see Figure 1). At the end of the process, the estimated volume is represented by an optimized number of elements in the mesh, what drastically reduces the size of the reconstructed data and consequently turns down the problems of memory storage.

Among the three stages, only the reconstruction one is here concerned with a GPU implementation (through both the projection and the backprojection operators) because of its high time consuming and its large memory load. By the way, classical algorithms with a regular parallelism (independent nested loops) can be used for the two other stages.

In this work, the $h_{i,j}$ elements of matrix H represent the intercepted length of the X-ray beam i with the mesh element j (see Figure 2). Contrary to the conventional case of a fixed grid (common voxel discretization), the $h_{i,j}$ elements can not be stored throughout the process of reconstruction because each time the mesh is modified the elements of the matrix H have to be computed.

IV. Implementation of the projection and the backprojection operators on GPU

Both the projection (Algorithm 1) and the backprojection (Algorithm 2) operators were implemented on GPU using a ray-tracing method. Three methods designed for 3D

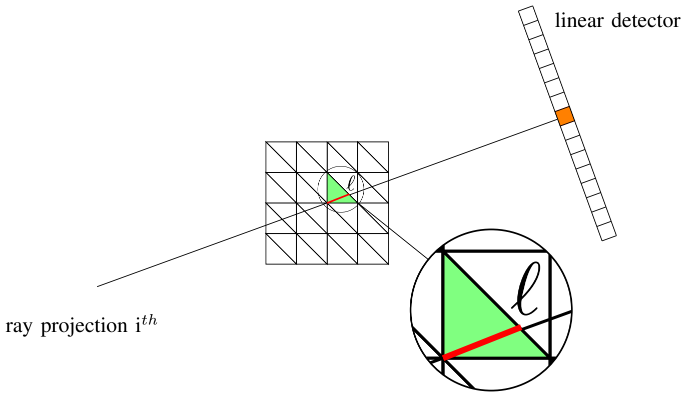


Figure 2: Example of the 2D projection operator.

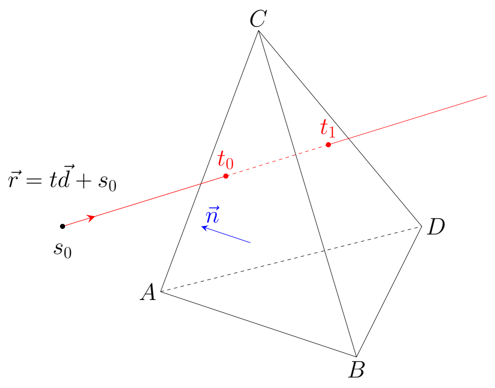


Figure 3: Ray-tracing by comparison of the parameter

rendering in computer graphics and adapted for our application were studied: one classification method (REFERENCE !!) and two parametric ones (REFERENCES !!). The first results have shown that one of the two parametric methods do not converge using single-precision floating point values because of the precision required to compute the intersection ray-tetrahedron. Also the classification method is slowest one. So, in the case of single-precision floating point, the second parametric method shows the best performances. The choice of single-precision processing is important as the ratio of performances can be very different according the used GPU. For example, there is a factor equal to 3 for the last Kepler GPU generation. The second parametric method thus used (REFERENCE!!) is based on the method of comparison of the parameters with the storage of the plane equations of the faces of the tetrahedra (see Figure 3):

- compute the normal vector $\vec{n}$ of the face, for example $\vec{AB} \times \vec{BC}$
- compute $s_1 = \vec{BD} \cdot \vec{n}$ and $s_2 = \vec{d} \cdot \vec{n}$
  if s_1 and s_2 have opposed signs, the ray $\vec{r}$ goes out of this face, and thus parameter $t = \dfrac{\vec{s_0B} \cdot \vec{n}}{\vec{d} \cdot \vec{n}}$

The outgoing face is identified by $t = \min\{t_i\}$ and the intercepted length is computed as $\ell = t_1 - t_0$.

The used 3D data structures are the following ones:

- i^{th} vertex: vertices$[3 \cdot i, \cdots, 3 \cdot i + 2]$
- vertex indexes of i^{th} tetrahedron: tetrahedron_corners$[4 \cdot i, \cdots, 4 \cdot i + 3]$
- neighbor opposite to corner of overall index j: tetrahedron_neighbors[j]
- uniform density of i^{th} tetrahedron: tetrahedron_densities[i]

Algorithm 1 Projection operator

```
1:  for all $n$ : $1 \to N_{rays}$ do
2:      $ray \leftarrow Ray(n)$ // read the coordinates of the ray
3:      $tetra \leftarrow Tetras_{initial}(n)$ // read the entry tetrahedron
4:      $t_{old} \leftarrow param\_ent(ray, next)$
5:      $intes \leftarrow 0$
6:      while $(tetra! = -1)$ do
7:          $den \leftarrow Dens(tetra)$ // read the density value
8:          $t \leftarrow param(ray, tetra)$ // compute the parameter of the next tetrahedron
9:          $intes \leftarrow intes + (t - t_{old}) \times den$ // weight the intercepted length by the density value and update the projection value
10:         $t_{old} \leftarrow p$
11:     end while
12:     $projection(n) \leftarrow intes$ // write the results
13: end for
```

Algorithm 2 Backprojection operator

```
1:  for all $n$ : $1 \to N_{rays}$ do
2:      $ray \leftarrow Ray(n)$ // read the coordinates of the ray
3:      $tetra \leftarrow Tetras_{initial}(n)$ // read the entry tetrahedron
4:      $t_{old} \leftarrow param\_ent(ray, next)$
5:      $intes \leftarrow projection(n)$
6:      while $(tetra! = -1)$ do
7:          $node \leftarrow tetra$
8:          $t \leftarrow param(ray, tetra)$ // compute the parameter of the next tetrahedron
9:          $Dens(node) \leftarrow Dens(node) + (t - t_{old}) \times intes$ // weight the intercepted length by the projection bin value
10:         $t_{old} \leftarrow t$
11:     end while
12: end for
```

This method requires a initialization stage of the rays to identify the input tetrahedron into the tetrahedral volume. This stage is very expensive in terms of time consuming when compared to a CPU execution time of one projection. In order to speed up this stage of initialization, a quad-tree hierarchical structure was implemented on each face of the volume. A regular tetrahedral volume is then considered, and a leaf of the quadtree only contains the index of two tetrahedrons (see Figure 4).

To optimize the performance of the quad-tree hierarchical structure, the following subdivisions of the faces of the tetrahedral volume were used:

- 16 x 16 (512 peripheral tetrahedrons)
- 32 x 32 (2048 peripheral tetrahedrons)

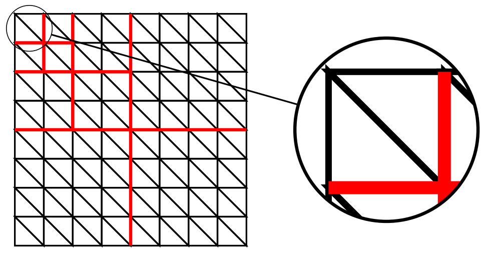


Figure 4: Decomposition of the surface of the tetrahedral volume.

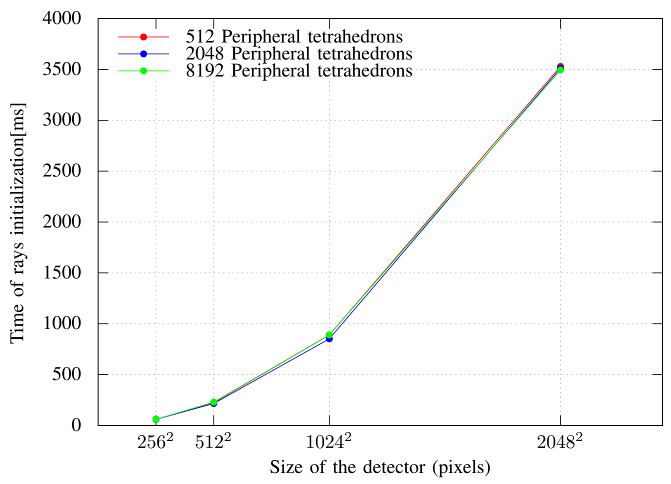


Figure 5: Execution time of the stage of initialization of the rays using a hierarchical structure.

- 64 x 64 (8192 peripheral tetrahedrons)

We observe that the execution time only depends on the number of rays to initialize (see Figure 5). The stage of initialization of the rays with a hierarchical structure is 700× faster than without (see Figure 6). This optimized initialization step is thus now less time consuming than the projection itself.

V. Results

Both the projection and backprojection algorithms were executed using 2D thread blocks. The size of the thread blocks was chosen in order to maximize the number of threads with respect to the material resources (registers) of the graphic card.

Each thread reads the first tetrahedron vector by its ID: if the value is equal to -1 the execution stops, otherwise the operator is performed. The latter is achieved when the next index of the tetrahedron is equal to -1.

A. Validation of the 3D projection operator

The validation of the projector was performed considering the object described in Table II (see Figure 7). A regular object was chosen to be perfectly described by the mesh so that it has no influence on the results. The results are shown in Figure 8. The 3D tetrahedral mesh is generated by using the library

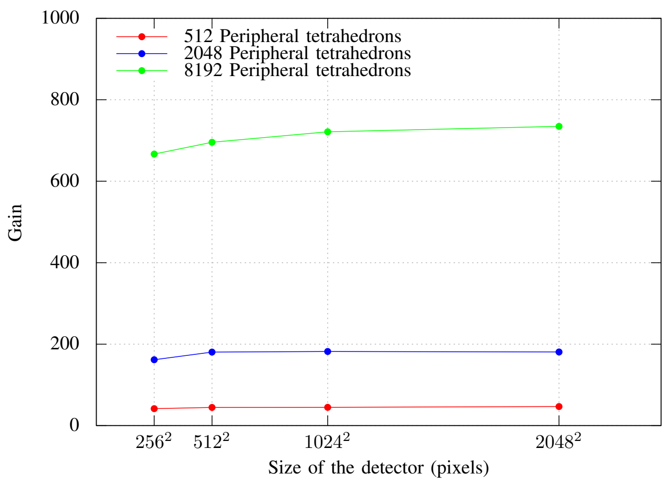


Figure 6: Gain between the stage of initialization without and with the hierarchical data structure.

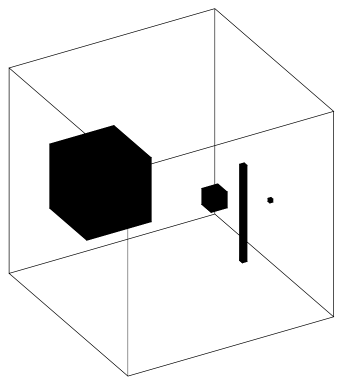


Figure 7: Object described to validate the 3D projector

Table II: Description of the test object used for the validation of the projector

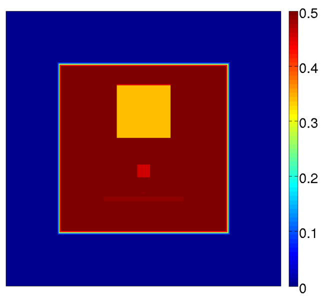
Object	x_{min}, x_{max}	y_{min}, y_{max}	z_{min}, z_{max}
Externe cube	-0.25, 0.25	-0.25, 0.25	-0.25, 0.25
1st hole	-0.1875, -0.03125	-0.1875, -0.03125	-0.078125, 0.078125
2nd hole	0.046875, 0.085938	0.046875, 0.085938	-0.019531, 0.019531
3th hole	0.082031, 0.09375	0.144531, 0.15625	-0.117188, 0.117188
4th hole	0.160153, 0.167969	0.128906, 0.136719	-0.003906, 0.003906

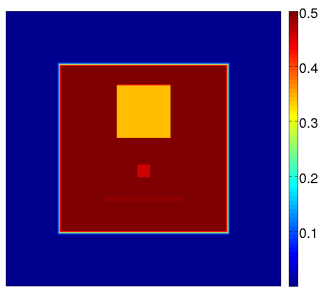
Tetgen [4]. The configuration of acquisition is described in Table III.

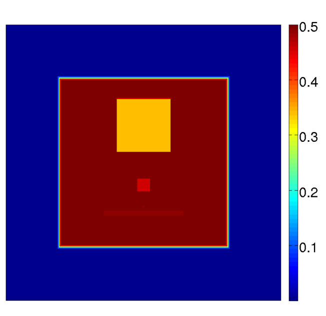
We studied the error introduced by a mesh which not perfectly describes an object. The test object is here an ellipsoid (see Figure 9) whose surface is discretized by a increasing number of points from 867 to 1622456. The error, expressed in percentage, is defined as:

$$Err = \frac{p_T - p_A}{p_A} \cdot 100\% \tag{2}$$

where, p_T is the vector of the tetrahedral projections and p_A is the vector of the analytical projections. Figure 10 shows the error, Err, according to the number of points describing the


(a) Analytical projection


(b) Tetrahedron projection CPU


(c) Tetrahedral projection GPU

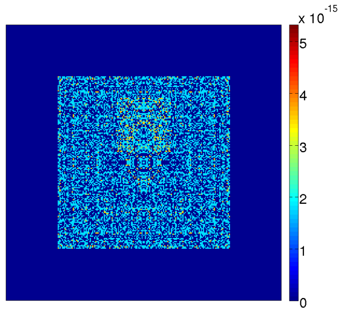

(d) Error between the projections (b) and (c)

Figure 8: Projections of the objet at $\theta = 0$ using (a) an analytical projector (intersection equations of the rays and the cubic elements) and a tetraedral projector (b) implemented on CPU and (c) implemented on GPU. The maximun error is about 10^{-15} (d).

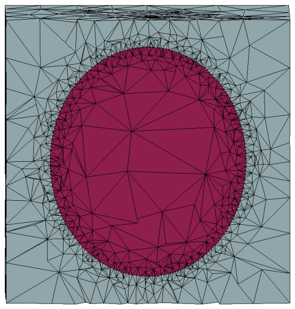


Figure 9: Ellipsoid surface described by 14011 points.

surface of the ellipsoid. We observe that the error decreases when the number of points describing ellipsoid increases, up to 357612 points. After what, the error remains constant at 1%.

B. Performance CPU/GPU

The performance were estimated by calculating the projection of the numerical phantom previously described (see Figure 7). The hardware used for these experiments are:

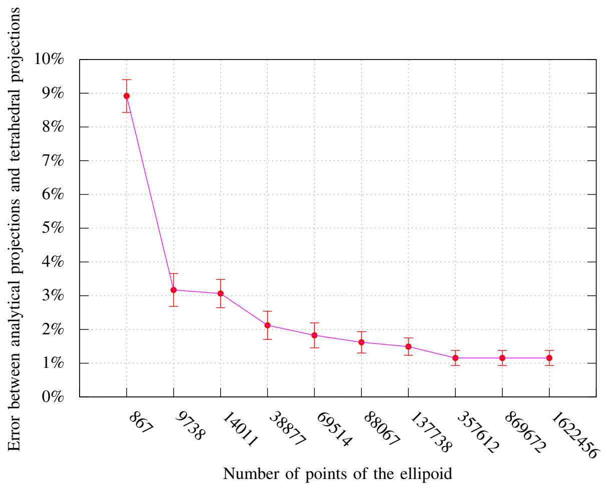


Figure 10: Error between the analytical projections and the tetrahedral ones with respect to the density of the tetrahedral mesh.

Table III: Configuration used for the validation of the projector on GPU.

Parameter	Value
source-object distance	98 mm
object-detector distance	132 mm
size of the detector	14.2mm $\times$ 14.2 mm
resolution of the detector (pixels)	256 $\times$ 256
number of the projections	64

- CPU: Intel Xenon E5440 @ 2.83GHz
- GPU: Nvidia-Fermi Tesla C2070

To characterize the performance obtained on GPU, comparisons were leaden by considering the execution time according to the detector size and the number of tetrahedra (see Figure 11). The memory transfert was taken into account to compute the speed-up factor. As result, the speed-up factor is in the range of 40$\times$ to 80$\times$. It depends on the size of the detector and the number of tetrahedra discretizing the volume. Large detectors are needed to take benefit from a high degree of parallelism. However, the sequential process of the tetrahedra limits the benefit.

We studied the influence of the memory transfers on the execution time of the projector implemented on GPU. Figure 12 shows that bigger is the detector, more time consuming are the memory transfers. The memory transfer time reaches the same value as the time of execution of the projection for a detector size of 8192$\times$8182 pixels. This is a main drawback that will impact speedup for larger detectors.

VI. Conclusion

In this paper we explored the feasibility of using Graphic Processor Units (GPUs) for tomographic reconstruction using tetrahedral discretization of the volume. We demonstrated fast projection/backprojection operators using a ray-tracing

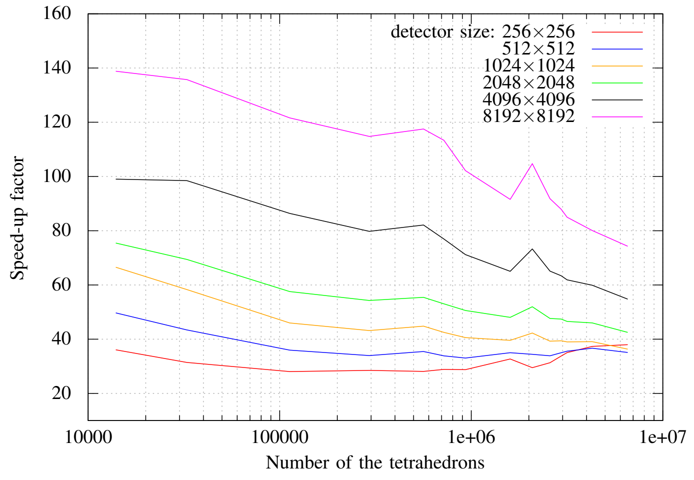


Figure 11: Speed-up of the GPU projector with respect to the number of the tetrahedral elements and considering several sizes of detector.

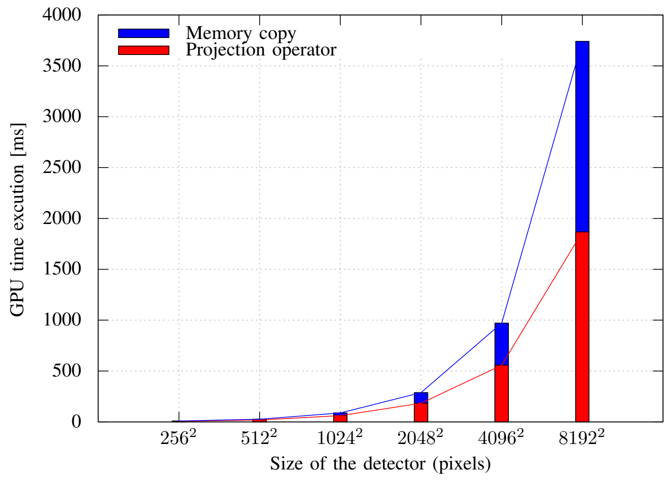


Figure 12: Execution time of the GPU projection vs the memory transfer.

algorithm with low required memory size. We also demonstrated a speedup factor between 40× and 80× in the case of an implementation of the operators on GPU over the CPU implementation with a low impact on the precision (error less than 1%).

Compared with a classical voxel-based method, the ATM method implemented on GPU suffers from a loss of memory locality and element regularity. Indeed, the voxel-based approaches discretize the volume into a set of regular elements, and GPU implementations can take benefit of that by using for example the texture memory to get data locality that results in divergence free and bank conflict free memory operations. Besides, all the ray beams across the same number of voxels that balances the work load among all the threads in the warp [5].

In our sutdy, an irregular discretization of the volume was chosen in order to fit the objet shape with a tetrahedral mesh. Because of the irregularity of the elements, two main disadavantages limit a highyield informatic optimization : the locality of the memory is lost (or almost), and the number of the tetrahedron processed by each thread belonging to the same warp can be different. That induces an unbalanced work load over a warp which means that threads still run while others remain idle. The minimization of the unbalanced work load is an important perspective to this work.

Acknowledgment

The authors would like to thank Tomacz Toczek for design the ray-tracing algorithm on the CPU/GPU in the framework of the 3D rendering in computer graphic.

References

[1] L. Herraiz, S. España, J. J. Vaquero, M. Desco, and J. M. Udías, “FIRST: Fast iterative reconstruction software for (PET) tomography,” *Physics in Medicine and Biology*, vol. 51, pp. 4547–4565, 2006.

[2] W. Zhuang, S. S. Gopal, and T. J. Herbet, “Numerical evaluation of methods for computing tomographic projections,” *IEEE Transaction on Nuclear Science*, vol. 41, no. 4, pp. 1660–1665, 1994.

[3] R. L. Siddon, “Fast calculation of the exact radiological path for a three-dimensional CT array,” *IEEE Transaction on Medical Imaging*, vol. 12, no. 2, pp. 252–255, 1985.

[4] H. Si, “Tetgen: A quality tetrahedral mesh generator and a 3D Delaunay triangulator,” http://www.tetgen.org, 2009.

[5] C.-Y. Chou, Y.-Y. Chuo, Y. Hung, and W. 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.