Fast System Matrix Generation on a GPU Cluster

Balázs Tóth, Milán Magdics, and László Szirmay-Kalos
Budapest University of Technology
http://www.iit.bme.hu/

***Abstract*—This paper presents an algorithm for Positron Emission Tomography reconstruction running on a GPU cluster. The most computation intensive part of the reconstruction process, the forward projection, is re-interpreted as a geometric problem, that can efficiently be solved by the graphics hardware. We also investigate the possibilities to further increase the speed and to sidestep the texture memory limitations by using not a single GPU, but a cluster of GPUs. To do so, the iteration scheme is modified to minimize the communication need between the GPU nodes.**

I. Introduction

In positron emission tomography we need to find the locations of positron–electron collisions. As a result of such collision, two gamma-photons are born that leave the collision location at two opposite directions [Gea07]. Photons may be absorbed by detectors forming grids. During measurement we collect the number of simultaneous photon incidents in detector pairs, also called *Lines Of Responses* or *LORs* (Fig. 1).

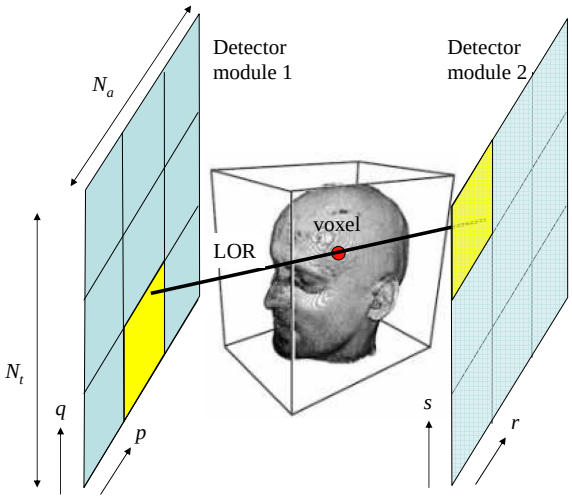

Fig. 1. Positron Emission Tomography. The pair of photons emitted at a voxel is detected by a pair of detectors. Detectors are on detector module planes. These planes are not necessarily parallel.

Let us denote the intensity of photon pair emissions in the voxels of a voxel grid of the measured object by $x_1, x_2, \ldots, x_{N_{voxel}}$, and the number of photon incidents in LORs by $y_1, y_2, \ldots, y_{N_{LOR}}$. The system response is characterized by a *system matrix* $\mathbf{A}$, where element $\mathbf{A}_{lv}$ defines the probability that a photon-pair emitted in voxel v is detected by LOR l [JSC+97]. The system matrix also defines the correspondence between voxel intensities $\mathbf{x} = (x_1, x_2, \ldots, x_{N_{voxel}})$ and measured values $\mathbf{y} = (y_1, y_2, \ldots, y_{N_{LOR}})$:

$$\mathbf{y} = \mathbf{A} \cdot \mathbf{x}.$$

The task of the reconstruction is to find voxel intensities of $\mathbf{x}$ based on the LOR incidents in $\mathbf{y}$, i.e. the solution of this equation. However, the matrix of this equation is not quadratic and is huge since the typical voxel number is 10^6 and the LOR number is about 10^7. Such equations can be attacked by iterative approaches, that iterate *forward-projection*,

$$\tilde{\mathbf{y}} = \mathbf{A} \cdot \mathbf{x}^{(n)}, \tag{1}$$

or in scalar form

$$\tilde{y}_l = \sum_{v'=1}^{N_{voxel}} \mathbf{A}_{lv'} x_{v'}^{(n)}, \tag{2}$$

and check whether or not the resulting guess $\tilde{\mathbf{y}}$ is close to the measured $\mathbf{y}$, and determine the next $\mathbf{x}^{(n+1)}$ from $\mathbf{x}^{(n)}$ accordingly.

For example, based on expected value maximization [SV82], we obtain the following correction, also called *back-projection* scheme:

$$\frac{x_v^{(n+1)}}{x_v^{(n)}} = \frac{1}{\sum_{L=1}^{N_{LOR}} \mathbf{A}_{lv}} \cdot \sum_{l=1}^{N_{LOR}} \mathbf{A}_{lv} \frac{y_l}{\tilde{y}_l} \tag{3}$$

Note that the equations of forward (equation 2) and back-projection (equation 3) are similar in the way that they take many known values (voxel intensities and LORs, respectively) and compute many unknown values (again, LORs and voxel intensities, respectively). This kind of "many to many" computation can be organized in two different ways. We can take known values one-by-one, obtain the contribution of a single known value to all of the unknowns, and accumulate the contributions as different known values are visited. We call this scheme *shooting*. The orthogonal approach would take unknown values (i.e. equations) one-by-one, and obtain the contribution of all known values to this particular unknown value. This approach is called *gathering*.

As our reconstruction scheme consists of a forward- and a back-projection steps, there are four different ways to implement the method depending on whether shooting or gathering approach is followed in forward- and back-projections. We emphasize that the distinction of these cases might be just the order of loops in a CPU implementation, but is a crucial design decision when the algorithm is run on the GPU since it defines

which loop is executed in parallel on the shader processors [SKSS08]. A shooting type forward-projection takes voxels one-by-one and indentifies those LORs that can detect this voxel. A gathering type forward-projection visits LORs one-by-one and finds those voxels that may contribute to this LOR. A shooting type back-projection, on the other hand, takes LORs and obtains the correction for those voxels that can be measured by this LOR. Finally, a gathering type back-projection visits voxels one-by-one, and identifies those LORs that measure this voxel.

In this paper, we propose the combination of a shooting type forward-projection and a gathering type back-projection. This way, both forward- and back-projections use the same elementary operation, the identification of those LORs that may measure a particular voxel. We call this elementary operation *voxel to LOR correspondence* determination.

Taking a geometric point of view and ignoring photon scattering, a LOR may measure a voxel if there exist lines crossing both detector surfaces and the voxel. This is not the case for most of the LOR and voxel combinations, thus the system matrix is sparse. Thus, the computation algorithm should consider this sparseness and should not waste time for trying to obtain zero elements.

The non-zero system matrix elements represent a probability, which can be built from elementary probability densities. Let $P(l|\vec{v},\vec{\omega})$ denote the conditional probability density that a photon-pair arrives at the two detectors of LOR l, provided that it is emitted in point $\vec{v}$ at directions $\vec{\omega}$ and $-\vec{\omega}$, respectively. Using this density we can establish a probability that a photon pair emitted in voxel v is detected in LOR l:

$$\mathbf{A}_{lv} = P(l|v) = \int_V \int_{2\pi} P(l|\vec{v},\vec{\omega}) \frac{\mathrm{d}\omega}{2\pi} \frac{\mathrm{d}v}{V}. \tag{4}$$

where V is the volume of the voxel. The volumetric integral is approximated by taking uniformly distributed points $\vec{v}_1,\ldots,\vec{v}_N$ in the voxel:

$$\mathbf{A}_{lv} \approx \frac{1}{N} \sum_{i=1}^{N} \int_{2\pi} P(l|\vec{v}_i,\vec{\omega}) \frac{\mathrm{d}\omega}{2\pi},$$

thus we need to find efficient methods for the computation of probability

$$P(l|\vec{v}) = \int_{2\pi} P(l|\vec{v},\vec{\omega}) \frac{\mathrm{d}\omega}{2\pi}. \tag{5}$$

The system matrix elements can be computed by lines traced through the volume and intersecting the detectors [MDB+08], where the lines were generated by points sampled on the detectors.

This paper proposes an approach that exploits the computational power of a GPU cluster to provide the non-zero system matrix elements and calculates the forward projection. In order to make the approach appropriate for GPU implementation, we define a special line density that is appropriate for GPU computation. Unlike previous GPU approaches [NGH08], [BS06] to solve the reconstruction problem on the GPU, we implement all steps directly on the GPU, thus no CPU intervention is needed. Comparing to papers about general tomography solutions on the GPU [BSKK07], [XM07], the novelty of our approach is the extension onto a GPU cluster, which becomes necessary when a single GPU card cannot store all measured data.

II. Voxel to LOR correspondence computation

In the proposed algorithm, the system matrix is approximated by lines. The lines are generated by the GPU, automatically assigning the workload to the parallel shader processors. To exploit GPU features, the system matrix integral is given a geometric interpretation. Let us consider the two detector modules. The module that is farther to point-like voxel $\vec{v}$ is called *primary*, the other is called *secondary*. Detectors of a module form a 2D grid. Let us assign indices p,q to the primary detector and indices r,s to the secondary (Fig. 1). Thus, LOR l gets four indices, (p,q,r,s). The integral in probability $P(l|\vec{v})$ (equation 5) is estimated by tracing lines that intersect both detector p,q and detector r,s and cross point $\vec{v}$. These lines are produced on the GPU by rasterizing the quad of the secondary detector module having set the eye position to point $\vec{v}$ and the camera window to the primary module.

With these lines we estimate an integral:

$$P(l|\vec{v}) \approx \sum_{j=1}^{M} P(l|\vec{v},\vec{\omega}_j) \frac{\Delta\omega_j}{2\pi}.$$

where M is the number of lines intersecting the primary detector, and line j has place vector $\vec{v}$ and direction vector $\vec{\omega}_j$ and represents solid angle $\Delta\omega_j$.

Lines are generated by rasterizing a quad and weights $P(l|\vec{v},\vec{\omega}_j)\Delta\omega_j/(2\pi)$ are obtained in the pixel shader.

A. Setting the virtual camera

The viewport resolution is set to be R times the resolution of the primary module, i.e. $RN_a \times RN_t$. This way, we trace $M = R^2$ lines through a primary detector.

The *camera transformation* is a concatenation of the following transformations (Fig. 2):

1) A translation that translates $\vec{v}$ to the origin.
2) A mirroring onto the origin that ensures that the secondary module will also be in front of the camera.
3) A rotation around axis x to align the normal vector of the detector ($\vec{n}$) with axis $-z$. The rotation angle is $\alpha = \mathrm{atan2}(n_y, -n_z)$.

This camera transformation also modifies the center of the primary detector module:

$$\vec{c}' = [\vec{c} - \vec{v}] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}.$$

The result of the camera transformation is shown by Fig. 3. The *perspective transformation* has two roles:

1) It first applies a *shearing* that is parallel to the x,y plane to ensure that the center of the detector $\vec{c}'$ is moved to

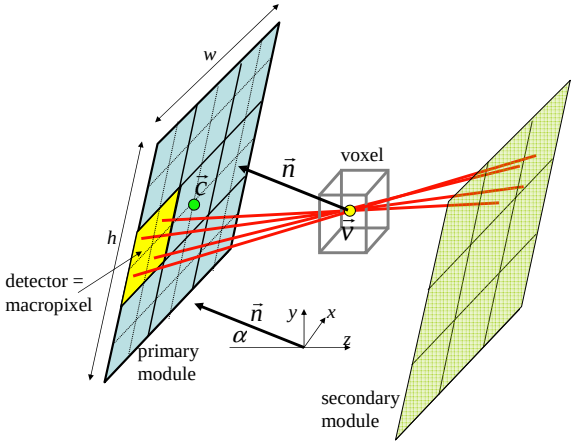

Fig. 2. Setting the camera. The eye position is the voxel and the window is the primary detector module.

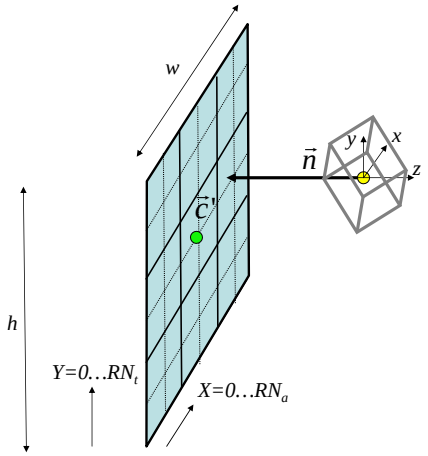

Fig. 3. Result of transforming to camera space.

axis z. The homogeneous linear transformation matrix of the shearing is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -c'_x/c'_z & -c'_y/c'_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2) Then, the eye position is mapped onto the ideal point of axis z, while projecting the visible frustrum to the cube of corners $(-1,-1,-1)$ and $(1,1,1)$:

$$\begin{bmatrix} -2c'_z/w & 0 & 0 & 0 \\ 0 & -2c'_z/h & 0 & 0 \\ 0 & 0 & -(f+b)/(b-f) & -1 \\ 0 & 0 & -2fb(b-f) & 0 \end{bmatrix},$$

where w and h are the width and the height of the primary detector module, respectively, f is the front clipping plane distance and b is the back clipping plane distance.

B. The pixel shader program

Having set the camera, the quad of the secondary module is sent down the rendering pipeline. The vertex shader transforms it to clipping space by executing both the camera and the perspective transformations, the rasterizer generates the pixels in its projection intepolating the pixel properties from the vertex properties, and the pixel shader processes pixels one-by-one. Note that a pixel of coordinates (X,Y) generated by the rasterization process corresponds to a line that crosses eye position $\vec{v}$ and the primary detector of $(p,q) = (X \text{ div } R, Y \text{div } R)$. The fractional part of $(X \bmod R, Y \bmod R)/R$, on the other hand, shows the location of the line–detector intersection on the detector surface. This information can be used to compute absorption in the material and to simulate physical processes inside the detector crystal. Pixel coordinate pair (X,Y) corresponds to camera space point

$$\vec{y}'_1 = \left(c'_x + w\left(\frac{X}{RN_a} - \frac{1}{2}\right), c'_y + h\left(\frac{Y}{RN_t} - \frac{1}{2}\right), c'_z\right).$$

In order to obtain world space intersection point $\vec{y}_1$, camera space point $\vec{y}'_1$should be rotated by $-\alpha$ around axis x and translated by $\vec{v}$, i.e. we should execute the inverse camera transform.

In order to identify the secondary detectors intersected by a line, we clear the viewport by zero and rasterize the secondary detector using Gouraud shading. Let us assign "colors"

$$(1,1),\ (1,N_t+1),\ (N_a+1,N_t+1),\ (N_a+1,1)$$

with red (R) and green (G) channels to the vertices of the quad of the secondary detector module. Adding 1 is necessary to distinguish detectors from the background, i.e. where the module is not visible. If the GPU interpolates color with perspective correction, then from the interpolated (R,G) pair we can obtain the indices of the secondary module as taking its integer part $r = (\text{int})R - 1$, $s = (\text{int})G - 1$. The fractional part again tells us where the intersection happened inside the detector. Similarly, we can assign the world space coordinates of the vertices to a texture coordinate register, thus the interpolation provides us with hit point $\vec{y}_2$ on the secondary detector.

As a result of the rasterization and interpolation, the pixel shader gets the target pixel coordinate from which indices (p,q) and world space hit point $\vec{y}_1$ on the primary detector can be obtained, colors (R,G) from which the secondary detector can be determined, and interpolated texture coordinates encoding hit point $\vec{y}_2$. Direction vector $\vec{\omega}$ of the line can be computed from these data. Weight a of this line will be the product of $P(l|\vec{v},\vec{\omega}_j)$, which is the probability that the photon pair reached the detectors, and $\Delta\omega_j$, is the solid angle represented by this line.

The line of the LOR has equation

$$\vec{y}(t) = \vec{y}_1 t + \vec{y}_2(1-t),\ t \in [0,1].$$

The probability that none of the two photons is absorbed is

$$P(l|\vec{v},\vec{\omega}_j) = \exp\left(-\int_0^1 \sigma_t(\vec{y}(t))\mathrm{d}t\right),$$

where $\sigma_t(\vec{y})$ is the extinction coefficient of the measured object at point $\vec{y}$. This integral can be estimated by *ray marching*, i.e. by taking K samples on the ray and using a simple quadrature:

$$P(l|\vec{v},\vec{\omega}_j) \approx \exp\left(-\sum_{k=0}^{K} \sigma_t(\vec{y}(k/K))|\vec{y}_1 - \vec{y}_2|/K\right).$$

Concerning solid angle $\Delta\omega_j$, we should note that the proposed rasterization process generates lines with non-uniform directional distribution, thus the integrand should be weighted accordingly (Fig. 4).

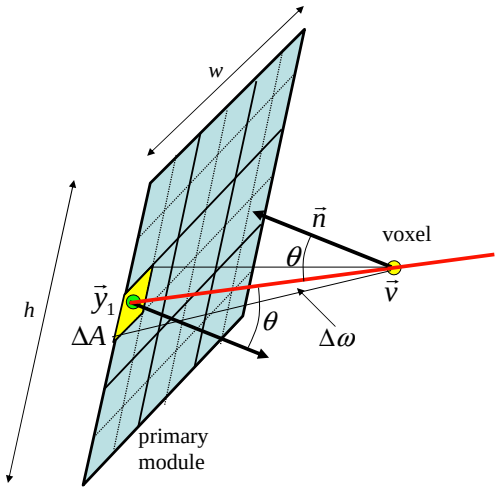

Fig. 4. Solid angle $\Delta\omega$ corresponding to a single line.

A pixel of the camera window represents $\Delta A = (w \times h)/(N_a \times N_t)/R^2$ area, which is seen from point $\vec{v}$ in solid angle

$$\Delta\omega = \frac{\Delta A \cos\theta}{|\vec{y}_1 - \vec{v}|^2} = \frac{wh\cos\theta}{N_a N_t R^2 |\vec{y}_1 - \vec{v}|^2}$$

where θ is the angle between viewing direction $(\vec{y}_1 - \vec{v})$ and the surface normal $\vec{n}$.

As a result, the pixel shader outputs (r, s, a) triplets as colors, encoding the hit point location on the secondary detector and the total weight of this line. The render target image is called the *correspondence image*.

C. Interpretation of the correspondence image

The pixels of the resulting high-resolution correspondence image encode lines crossing point $\vec{v}$. From another point of view, the image is a representation of a row of the sparse system matrix. If we need a matrix element of this voxel and LOR of detector p, q, then those pixels should be visited where the coordinates are

$$(pR, qR),\ \ldots,\ ((p+1)R-1,\ (q+1)R-1).$$

These pixels form a *macropixel* (Fig. 2). In order to obtain a LOR of detector p, q and detector r, s, those pixels should be extracted from macropixel p, q where the color is r, s, and the weights of such pixels are added.

III. Forward-projection

Note that the correspondence image represents the effect of a single voxel (more precisely a single sample of a single voxel) onto all LORs. Thus, in order to execute the forward projection (equation 1) all voxels should be processed and the weights should be added.

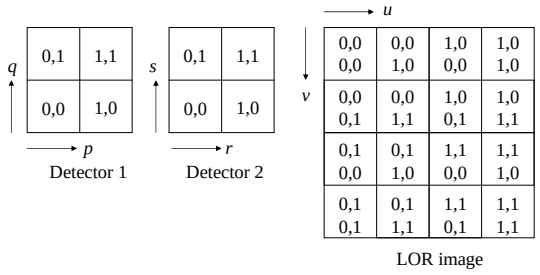

Fig. 5. The LOR image. In this figure we assumed that a detector module has 2×2 resolution.

To support this operation, the LORs are encoded in a two-dimensional texture, called *LOR image* (Fig. 5). As a LOR is defined by four indices (p, q, r, s) but the graphics hardware allows only two-dimensional textures to be render targets, the LORs are given as a two-dimensional texture atlas, where a texel of coordinates (u, v) represents that LOR, for which:

$$u = \frac{pN_a + r}{N_a N_t},\ \ v = \frac{qN_t + s}{N_a N_t}.$$

The rendering pass that adds the contribution of the current voxel sample to the LOR texture renders a single quad that covers all target texels. We use the *vertex shader* to visit all pixels of the correspondence image and to determine the LORs to which this contribution should be accumulated. The vertex shader computes the u, v coordinates of a LOR from the four LOR indices, transforms it to homogeneous clipping space as $((1+u)/2, (1-v)/2, 0, 1)$ and emits the point primitive with this position and its summed weight multiplied by voxel contribution x_v/N. Setting alpha blending to add colors, this contribution will be added to the respective LOR.

IV. Back-projection

Our back-projection algorithm is based on the concept of gathering, i.e. it evaluates equation 3 one-by-one. For a particular equation, those LORs should be identified, which measure this voxel. Note that this task is solved by the *voxel to LOR correspondence computation* that is discussed in the previous section. The result of this step is the correspondence image. The difference between forward- and back-projections is the processing of the correspondence image. Now, we have to evaluate equation 3, that is, having the correspondence image

encoding system matrix elements $\mathbf{A}_{lv}$ for $l = 1 \dots N_{LOR}$ and for the current voxel v, we have to obtain two sums

$$\sum_{L=1}^{N_{LOR}} \mathbf{A}_{lv}, \qquad \sum_{l=1}^{N_{LOR}} \mathbf{A}_{lv} \frac{y_l}{\tilde{y}_l}.$$

A possible solution would be to render a single pixel viewport where the pixel shader would compute both sums and and would output either the two sums in two color channels or their ratio in a single color channel. However, this approach would not exploit the parallel GPU architecture. Thus, we reduce the resolution of the correspondence image gradually. In the first run, the resolution is not reduced to 1×1 only to the half of the original resolution, and we compute the above sums only for 4 pixels. Then in each consecutive rendering step, the resolution is halved and in each step, the pixel shader adds four pixels together.

V. Distributed reconstruction

The proposed iteration scheme generates a new LOR estimate in each iteration step as a result of a rendering to a texture. Thus LORs should be encoded in a texture. Thus, the maximum render target and texture resolutions, and ultimately the available texture memory impose limits on the number of LORs. A possible solution of this problem that does not sacrifice performance is the application of multiple GPU cards, or a GPU cluster. In this case, the task is to distribute the computation burden among them, with practically no extra communication since such communication is very costly on this hardware.

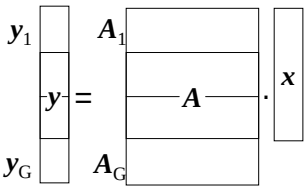

Fig. 6. Distribution of the system matrix and the LOR estimate to different GPUs.

We propose to assign just a subset of LORs to every GPU node of the available G nodes, and run the forward-projection and the back-projecting correction algorithm completely independently on these nodes. Mathematically, the LOR incidence vector $\mathbf{y}$ is broken down to vectors $\mathbf{y}_1, \dots, \mathbf{y}_G$ of smaller N_{LOR}/G dimension, and matrix $\mathbf{A}$ is also decomposed to G minor matrixes $\mathbf{A}_1, \dots \mathbf{A}_G$ of dimension $N_{LOR}/G \times N_{voxel}$ (Fig. 6). Thus, node g provides an independent estimate of the complete voxel intensity vector using partial LOR vector $\mathbf{y}_g$ and using minor matrix $\mathbf{A}_g$. These estimates are less accurate than the result that would be obtained by considering all LORs simultaneously. When these independent guesses are available, we composite these guesses together to obtain the high accuracy estimate. The final estimate is obtained as the combination of guesses $\mathbf{x}[g]$

$$\mathbf{x} = \sum_{g=1}^{G} \alpha_g \mathbf{x}[g], \qquad \sum_{g=1}^{G} \alpha_g = 1,$$

where $\alpha_1, \dots, \alpha_G$ are yet unknown weights, which should be selected to minimize the L_2 error

$$\epsilon = (\mathbf{y} - \mathbf{A} \cdot \mathbf{x})^2 = \left(\mathbf{y} - \mathbf{A} \cdot \sum_{g=1}^{G} \alpha_g \mathbf{x}[g] \right)^2.$$

Let us introduce the notation of the forward projection of guess $\mathbf{x}[g]$ by minor matrix $\mathbf{A}_h$ as

$$\tilde{\mathbf{y}}_{hg} = \mathbf{A}_h \cdot \mathbf{x}[g].$$

With this, the error is

$$\epsilon = \sum_{h=1}^{G} \left(\mathbf{y}_h - \sum_{g=1}^{G} \alpha_g \tilde{\mathbf{y}}_{hg} \right)^2.$$

To find the minimum, the partial derivatives according to α_g are made equal to zero, and constraint $\sum_{g=1}^{G} \alpha_g = 1$ is also taken into account:

$$\frac{\partial \epsilon}{\partial \alpha_k} = \sum_{h=1}^{G} 2 \left(\mathbf{y}_h - \sum_{g=1}^{G} \alpha_g \tilde{\mathbf{y}}_{hg} \right) \cdot \tilde{\mathbf{y}}_{hk} = 0.$$

This results in a linear equation for the unknown weights:

$$\mathbf{B} \cdot \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_G \end{bmatrix} = \mathbf{b}, \qquad \sum_{g=1}^{G} \alpha_g = 1,$$

where

$$\mathbf{B}_{kg} = \sum_{h=1}^{G} \tilde{\mathbf{y}}_{hg} \cdot \tilde{\mathbf{y}}_{hk}, \qquad \mathbf{b}_k = \sum_{h=1}^{G} \mathbf{y}_h \cdot \tilde{\mathbf{y}}_{hk}.$$

In our system, the number of GPUs is at most five, thus, the additional computational cost of combination is the solution of a linear equation with five unknowns, which is negligible compared to the other steps. Thus, the performance scales linearly in this range.

VI. Results

The system has been implemented on a 5 node HP Scalable Visualization Array (SVA), where each node is equipped with an NVIDIA GeForce 8800 GTX GPU, programmed under GLSL. The nodes are interconnected by Infiniband.

Fig. 7 depicts the original density field used as the extinction coefficient in the calculations and the reconstructed voxel intensity map encoded with false colors. The voxel array has 128^3 resolution.

Fig. 8 shows the correspondence images for two different voxel positions. Note that we used color scaling to map colors from the range $[1 \dots N_a + 1, 1 \dots N_t + 1]$ to $[0 \dots 1, 0 \dots 1]$. We assumed 32×32 resolution of the detector modules.

Fig. 9 contains the weights of LORs when only the geometry is considered and when the absorption of the voxels is also

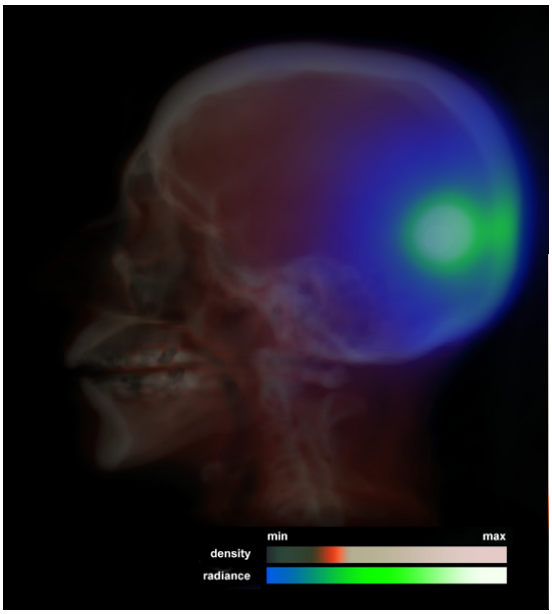

Fig. 7. Reconstructed volume composited with the density field (the extinction coefficient).

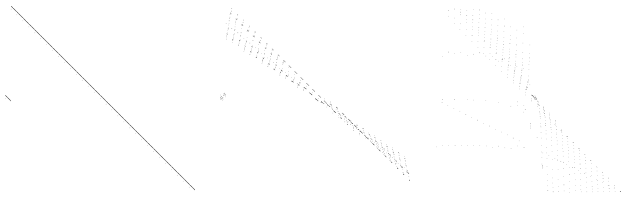

Fig. 8. Correspondence images rendered with false colors for three different voxels.

computed. In this experiment, we used a spherical numerical phantom.

Concerning the performance, on a single node the execution time of the correspondence image rendering is 0.06 msec when $R = 4$, a single LOR image generation of resolution $32^2 \times 32^2$ takes 0.07 msec, the complete forward-projection iteration lasts 0.13 msec for a single voxel and 4.5 minutes for all 128^3 voxels. The reduction of the correspondence image to a single scalar requires 0.04 msec, thus a single back-projection iteration for all voxels need 90 seconds. Due to a good scalability, 10 iterations and the composition of the result on the five node cluster is possible in less than 10 minutes.

VII. Conclusion

This paper proposed a GPU based algorithm for the reconstruction of PET measurements. The original approach is restructured to exploit the massively parallel nature of GPUs. We also applied a further parallelization to extend the algorithm to multiple GPUs taking into account the communication bottleneck between different GPU nodes. This approach can reduce the computation time of the reconstruction to a few minutes from hours of computation needed by CPU solutions.

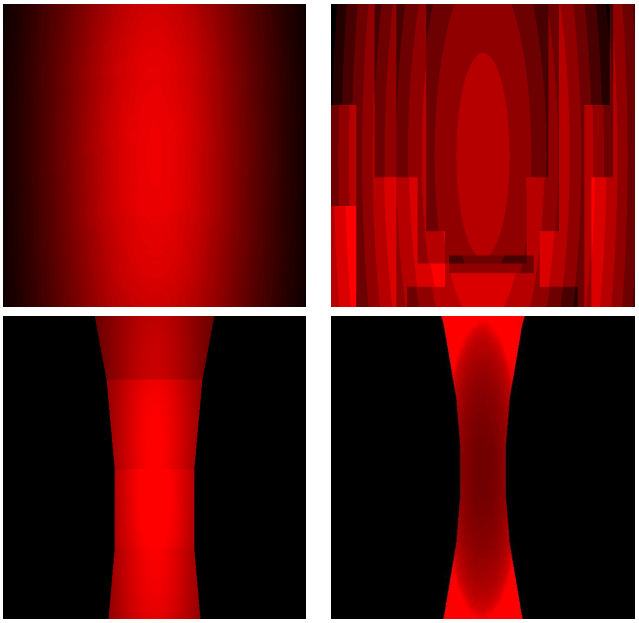

Fig. 9. Weights of LORs.

Acknowledgement

This work has been supported by the Terratomo project of the National Office for Research and Technology and by OTKA K-719922 (Hungary), and by the Croatian-Hungarian Action Fund.

References

[BS06] Bing Bai and Anne Smith. Fast 3D iterative reconstruction of PET images using PC graphics hardware. In *IEEE Nuclear Science Symposium*, pages 2787–2790, 2006.

[BSKK07] O. Bockenbach, S. Schuberth, M. Knaup, and M. Kachelriess. Image reconstruction platforms; state of the art, implications and compromises. In *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine — Volume 9*, pages 17–20, 2007.

[Gea07] Geant. Physics reference manual, geant4 9.1. Technical report, CERN, 2007.

[JSC+97] C. A. Johnson, J. Seidel, R. E. Carson, W. R. Gandler, A. Sofer, M. V. Green, and M. E. Daube-Witherspoon. Evaluation of 3D reconstruction algorithms for a small animal PET camera. *IEEE Transactions on Nuclear Science*, 44:1303–1308, June 1997.

[MDB+08] Sascha Moehrs, Michel Defrise, Nicola Belcari, Alberto Del Guerra, Antonietta Bartoli, Serena Fabbri, and Gianluigi Zanetti. Multi-ray-based system matrix generation for 3D PET reconstruction. *Phys. Med. Biol.*, 53:6925–6945, 2008.

[NGH08] Michel Desvignes Nicolas Gac, Stphane Mancini and Dominique Houzet. High speed 3D tomography on CPU, GPU, and FPGA. *EURASIP Journal on Embedded Systems*, 2008. Article ID 930250, 12 pages.

[SKSS08] L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.

[SV82] L. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Med. Imaging*, 1:113–122, 1982.

[XM07] F. Xu and K Mueller. Real-time 3d computed tomographic reconstruction using commodity graphics hardware. *Phys Med Biol*, pages 3405–3419, 2007.