

GRAPHICS HARDWARE ACCELERATED RECONSTRUCTION OF SPECT WITH A SLAT COLLIMATED STRIP DETECTOR

FOR THE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, COMPUTER VISION AND PATTERN RECOGNITION

J. De Beenhouwer, R. Van Holen, S. Vandenberghe, S. Staelens, Y. D'Asseler, I. Lemahieu

Ghent University, Elis Department, Sint-Pietersnieuwstraat 41 B-9000 Ghent, Belgium

ABSTRACT

3D iterative reconstruction of Single Photon Emission Computed Tomography (SPECT) acquisitions is a widely used but computationally intensive process. Reconstructing data from a rotating slat collimated system increases the computation time even further. Since the sensitivity profile of such a camera is not constant over the field of view, it is important to include a system model that is as accurate as possible. We derived a Monte Carlo model of the physics of the measurement process. The advances in programmable graphics hardware make it possible to efficiently model the 3D reconstruction algorithm on consumer grade graphics cards in order to reduce the computation time. This paper describes an implementation of the reconstruction algorithm which is as accurate as the software algorithm while still achieving a 10-fold speed increase over the optimized software version.

Key words: Iterative reconstruction, graphics hardware, SPECT

1. INTRODUCTION

3D iterative reconstruction of Single Photon Emission Computed Tomography (SPECT) is a widely used but computationally intensive process. Compared to analytical methods such as filtered backprojection, it can achieve higher image quality because accurate system models of the image formation process can be incorporated to enable quantitative image reconstruction, noise reduction and resolution recovery. However, these system models, often based on Monte Carlo simulations, increase the computation time even further. Classical SPECT setups with parallel hole collimators have the advantage of a constant sensitivity over the whole field of view. This is however no longer the case when using a rotating slat collimator where plane integrals of the activity distribution are measured as shown in figure 1. The detector consists of 192 detector elements along the x-axis.

The non-constant sensitivity profile of this type of detector was described in [1]. A 3D reconstruction method for this system is based on the classical Maximum Likelihood Expectation Maximization algorithm (MLEM) [2] and requires the use of an accurate system model. The MLEM principle is shown in equation 1 where λ_j^k describes the activity in voxel j at iteration step k . The parameter c_{ij} describes the system matrix, indicating the probability of detection of a photon from voxel j in detector element i . The system matrix was derived from a Monte Carlo simulation of a planar source vertically centered above the central detector element. Using the rotational symmetry of the acquisition it is possible to calculate all the elements of the system matrix during the reconstruction process from the sensitivity of the points in this plane [3]. The normalization image $\sum_i c_{ij}$ is calculated similarly. The iterative method is composed of a forward projection of the current image estimate, which is then compared to the measured projection q_i . The resulting ratio is then used as a measure for correcting the current estimate after the backprojection. As described in [1] the slat collimator also revolves around its own axis. This spin rotation introduces an extra computational burden as each image must be projected not only for each SPECT angle but also for each spin angle.

$$\lambda_j^{k+1} = \frac{\lambda_j^k}{\sum_i c_{ij}} \sum_{i=1}^C c_{ij} \frac{q_i}{\sum_{j'=1}^N c_{ij} \lambda_{j'}^{k'}} \quad (1)$$

In order to reduce the computation time we have developed an implementation of the reconstruction algorithm on graphics hardware. The previously fixed rendering pipeline of graphics processing units (GPU's) has turned into an almost fully programmable one, allowing it to be used for more general purpose computing. In [4] the acceleration of reconstruction algorithms such as the Simultaneous Algebraic Reconstruction Technique (SART) [5] and Ordered Subsets Expectation Maximization (OS-EM) [6] was investigated and a speedup of the order of 1-2 magnitudes was achieved with a level of quality comparable to software implementations. In [7] the same authors exploited 8-bit texture mapping facilities to accelerate 3D filtered backprojec-

Presenting author: Jan De Beenhouwer
Email: jan.debeenhouwer@ugent.be
Tel./Fax: +32 9 264 66 18

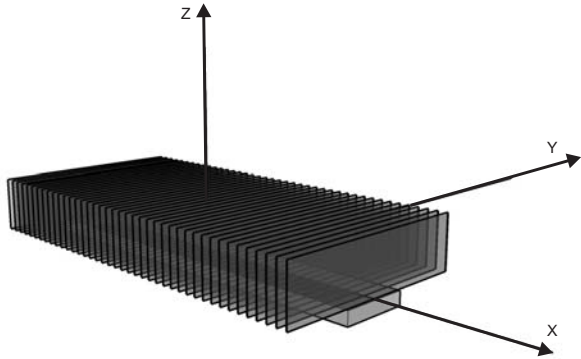


Fig. 1. A rotating slat collimator with strip detector

tion 37-fold, although with slightly more noisy results. In [8] a hardware-based acceleration of OS-EM was proposed specifically for varying focal-length fan-beam collimators (VFF). By taking advantage of the geometrical symmetry of the VFF point-spread function and by using Octree compression it was possible to reduce the matrix size and load it into video memory, achieving a 10-fold speedup. Although

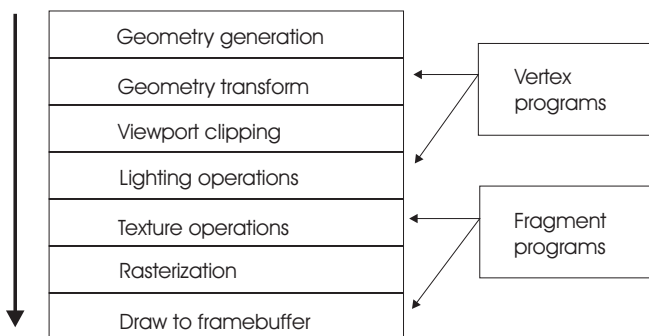


Fig. 2. The programmable OpenGL pipeline

it has been shown in these publications that a high speed increase can be achieved on graphics hardware, it is not achieved without any loss of accuracy. This loss of accuracy can be attributed to the fact that most GPU reconstruction algorithms use approximations to the software algorithm in order to achieve a maximum speed increase. The advantage of the use of a detailed system model required by a rotating slat collimator would however be partly undone by this loss of accuracy. In contrast with previous publications, this paper will therefore focus on the acceleration of the 3D reconstruction algorithm without making any compromise regarding accuracy when compared to the software version.

2. GPU STREAMING MODEL

Modern graphics hardware is capable of 32 bit floating point calculations throughout the GPU pipeline. Figure 2 shows a simplified version of the programmable graphics pipeline on modern GPU's. The GPU follows a stream architecture in which a constant stream of polygons (consisting of vertices) is sent through a vertex processor. The vertex processor transforms the vertex coordinates from a local to a world coordinate system, clips the polygons against a viewing volume and applies lighting operations on the vertices. After this stage the vector based geometry is rasterized into fragments. A fragment processor executes a fragment program which applies textures to the fragments and stores them as pixels in a framebuffer. All vertices and fragments can be processed independently and thus in parallel which makes the stream model very efficient. The GPU can be used as a General Purpose Graphics Processing Unit (GPGPU) by rendering a fullscreen polygon to an offscreen memory buffer resembling a floating point matrix. This way a fragment program will operate as a kernel on each fragment and thus on each element of that matrix.

3. METHODS

3.1. GPU implementation of MLEM

In order to implement an iterative algorithm it is necessary to create a feedback loop in order to access information from the previous iteration step. The image space is represented by a stack of 192 2D slices in the form of textures. In each texture 4 floating point values can be stored per pixel. Therefore the whole image space can be represented by 48 textures. Each value in a texture pixel represents a voxel in a slice of the image space. However, the GPU can not read from and write to the same texture at the same time. The solution exists in the form of two identical textures bound to a single framebuffer as shown in figure 3. The information from the previous iteration step is read from a texture slice (texture 1) bound as a read buffer while the output is written to a texture bound as a draw buffer (texture 2). After the completion of the step, the roles of texture 1 and 2 are swapped: texture 1 is bound as a draw buffer and texture 2 as a read buffer. This functionality has been introduced by the OpenGL framebuffer object extension. It enables a lightweight and more flexible memory model for GPU's that avoids costly context switches as only one framebuffer is necessary.

As described in [1], a pixel based sensitivity correction is necessary for our 3D reconstruction algorithm. The model was computed as explained in section 1 and is stored in a reduced system matrix (R) as shown in figure 4. It consists of 288 slices parallel to the detector surface at increasing distance (along the z-axis). Ideally this system model

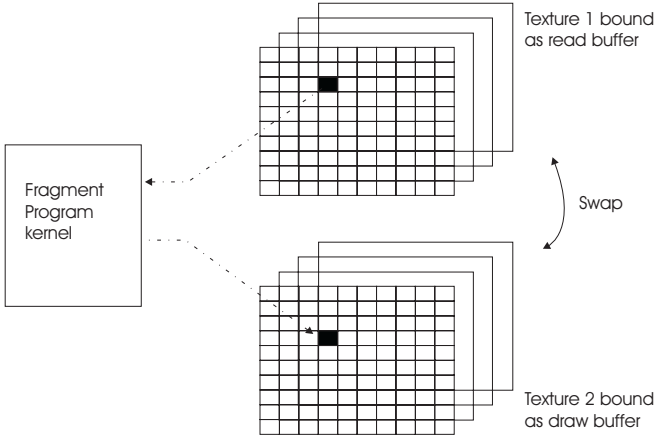


Fig. 3. Fragment program acting as a kernel on each pixel of a 2D texture bound to the drawable framebuffer. A feedback loop is established by swapping two identical textures and reading in the previous result as a normal texture.

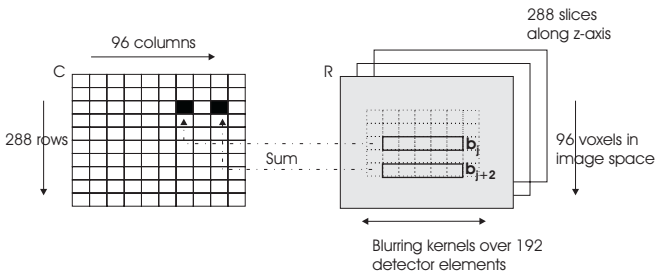


Fig. 4. Compressed system matrix C and reduced system matrix R. The row elements of R represent blurring kernels and are added and stored in the pixels of the compressed system matrix.

should describe the probability of detection c_{ij} of a photon from each voxel in each detector element. However, using symmetry it is sufficient to store this only for voxels in the plane perpendicular to the x-axis and centered over the detector. A transformation is then required in order to place the correct sensitivity profile at the current voxel. Figure 4 shows how the spatial blurring kernel b_j , which specifies the sensitivity profile of a voxel over the central detector element, is stored in all 288 slices. Using symmetry it is again sufficient to store the blurring kernel for the 96 voxel elements of the lines in the plane along the y-axis that cover half of the image space dimension. A condensed form is also calculated as matrix C, which adds all elements of each row together. Each row in C therefore represents a slice of R and thus a perpendicular distance to the detector. This condensed form is used to create the normalization image $\sum_i c_{ij}$ for each voxel in the image space as shown in figure

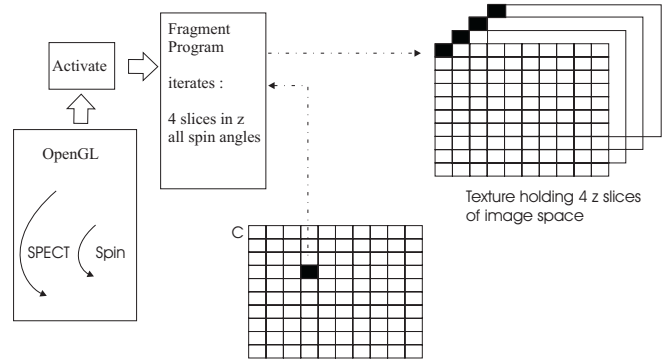


Fig. 5. A normalization map is calculated by a backprojection. A fragment program acts as a kernel looping over all (x,y) voxels in 4 z slices at a time.

5. This step is actually a backprojection, which loops over all voxels of the image space. For each SPECT angle, and for each spin angle over a certain voxel (x,y,z), the transformation is calculated which indexes into the compressed matrix C which contains the contribution to the voxel. This part of the algorithm is executed by a fragment program which loops over all spin angles for 4 z positions at the same time. Therefore 48 slices need to be rendered, each looping over all SPECT angles in OpenGL.

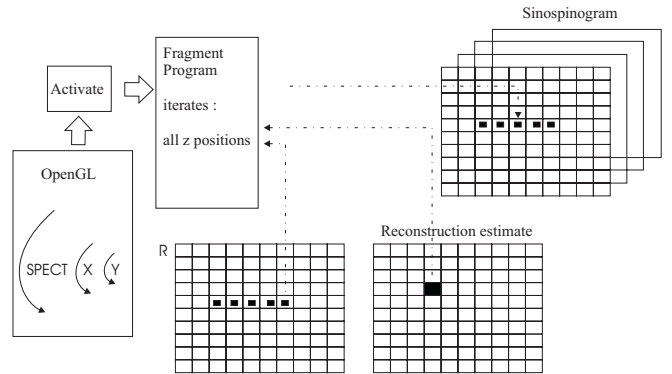


Fig. 6. The forward projection step to calculate a sinospinogram.

The forward projection of the MLEM algorithm is shown in 6. The projection is not a sinogram but a sinospinogram due to the extra rotation of detector around its own axis as explained in [1]. The OpenGL code loops over all (x,y) positions for each SPECT angle and activates a fragment program which loops over all z positions. Each element of the sinospinogram is calculated by spreading out each (x,y,z) value of the current image estimate by the corresponding blurring kernel in the reduced system matrix R. This is in

effect a "scatter operation" which is not directly possible on a GPU as each pixel writes to a predefined output position. This scatter operation was however easily converted into a gather operation as for each pixel in the sinospinogram it is possible to calculate which elements will contribute to it.

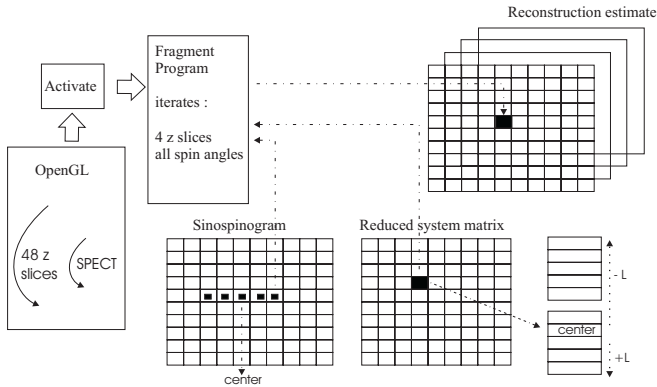


Fig. 7. The sinospinogram estimate is corrected by the sinospinogram measured by the acquisition and backprojected into the image space.

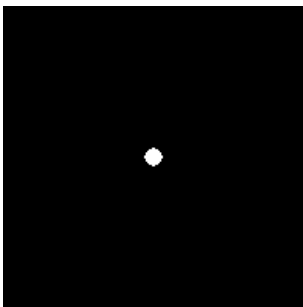


Fig. 8. The analytical cylindrical phantom (central transaxial slice)

The backprojection step is similar to the calculation of the normalization image as shown in figure 7. The OpenGL code loops over all SPECT angles and 48 z-slices, while the fragment program calculates 4 z positions at a time for every (x,y) pixel. The reduced system matrix is represented in a different ordering format to reduce memory bandwidth. The blurring kernel is now stored orthogonally into 2L slices, with 2L the width of the blurring kernel. For each voxel at a certain SPECT angle and a certain spin angle, a transformation is calculated to index into a row element of the sinospinogram obtained in the forward projection step. Each value around this central element of the row must then be multiplied by the corresponding element of the blurring kernel in the system matrix. The result is also

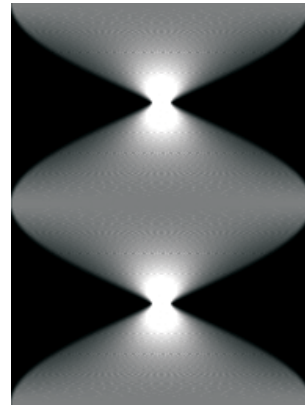


Fig. 9. The forward projection of the analytical phantom at the 30th SPECT angle.

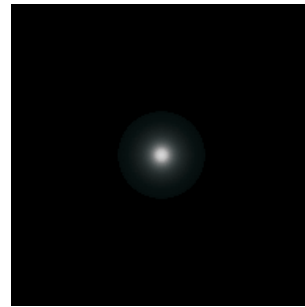


Fig. 10. Reconstruction of the cylindrical phantom at the 9th iteration (central transaxial slice).

updated by the measured sinospinogram. Both the measured and estimated sinospinogram are stored in the different color channels of the slices of the sinospinogram matrix in order to conserve memory bandwidth. Once the back-projection is calculated it is weighted by the normalization image before the next iteration step can occur.

3.2. Benchmark reconstructions

An analytical phantom in the form of a cylinder, shown in figure 8, was used to assess the image quality. The phantom was forward projected using the GPU for 60 SPECT angles and for each SPECT angle over 256 spin angles in order to obtain sinospinograms as shown in figure 9. Next, the sinospinograms were used for both the software and hardware reconstruction. The reconstruction volume consisted of 192 slices of 192x192 pixels. Both software and hardware reconstructions were run on an intel P4 2.8Ghz processor with an nvidia 6800 ultra GPU.

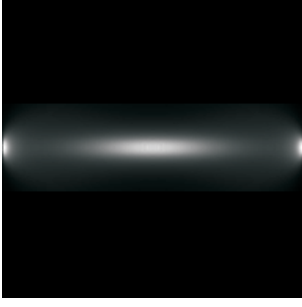


Fig. 11. Reconstruction of the phantom image at the 9th iteration (central sagittal slice).

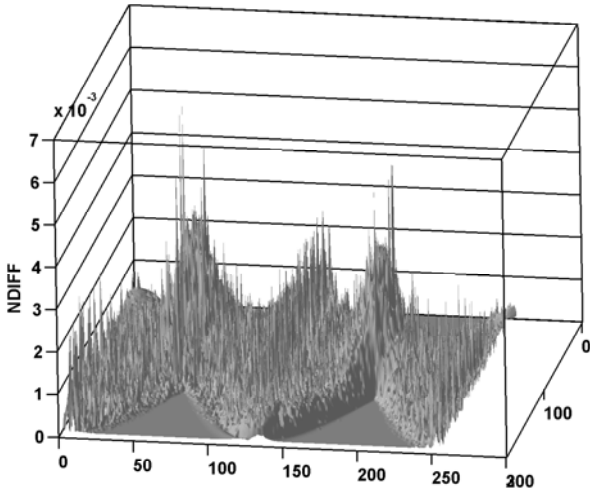


Fig. 12. Normalized difference image comparing the GPU and CPU forward projection at the 30th SPECT angle for the 9th iteration.

3.3. Assessment of image accuracy

The accuracy assessment of the GPU implementation consists of three parts. Firstly both the forward projections and the reconstructed images of the software and hardware algorithm are compared over nine iterations. The root mean squared (RMS) value for the reconstructed images was normalized by the mean software image value (μ_1) in a region of interest covering the actual calculation grid. The RMS value for the forward projected images was normalized by the mean software image value over the whole grid (μ_2). Both normalized RMS values (NRMS) are expressed in equation 2 where $s(k)$ and $h(k)$ represent pixel k of respectively the software and hardware reconstructions and

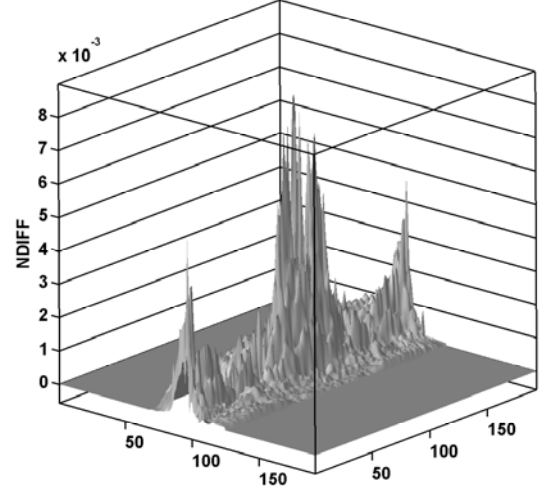


Fig. 13. Normalized difference image comparing the GPU and CPU reconstruction at the 9th iteration (central sagittal slice).

forward projections.

$$NRMS = \frac{\sqrt{\frac{\sum_1^N (s(k) - h(k))^2}{N-1}}}{\mu_{1,2}} \quad (2)$$

Secondly, both the software and hardware reconstructed images are compared separately to the analytical phantom. The mean squared error (MSE) and the peak signal to noise ratio (PSNR) are determined for both reconstructions over nine iterations. The MSE is shown in equation 3 where $a(k)$ represents a voxel value of the analytical phantom. The PSNR is defined by equation 4 where MAX_I is the maximum pixel value of the image. It serves as a measure of quality of the reconstruction, taking into account the power of corrupting noise through the MSE.

$$MSE = \frac{\sum_1^N (s, h(k) - a(k))^2}{N-1} \quad (3)$$

$$PSNR = 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \quad (4)$$

Finally, a difference image between software and hardware at the 9th iteration is generated. The difference images for both the forward projection and the reconstructed image provide a visual impression of the spread of the error and were normalized by their respective mean software value.

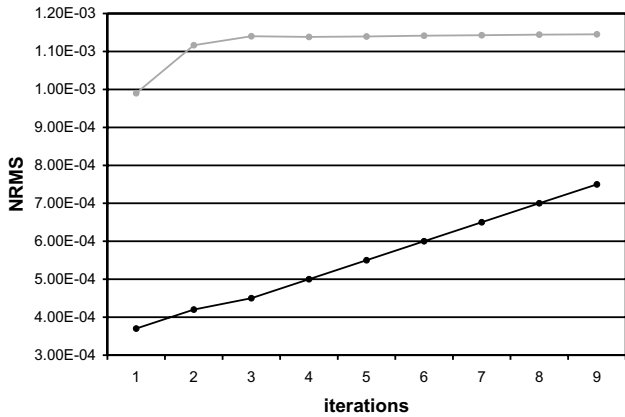


Fig. 14. The NRMS values over nine iterations for both the forward (grey) and backward (black) projection steps.

4. RESULTS

Figures 10 and 11 show a central transaxial and sagittal slice of the reconstruction at the 9th iteration for the GPU algorithm. The difference images in figures 12 and 13 provide an indication of the spread of the error between the software and hardware reconstruction. The normalized RMS value over nine iterations is shown in figure 14. The error remains under 8×10^{-4} for the backward projection and remains under 1.2×10^{-3} for the forward projection. The results for the comparison of the reconstructions with the analytical phantom are shown in figures 15 and 16. Both the MSE and the PSNR values for the GPU visually coincide with the software algorithm. The difference is plotted on the right axis and remains under 3×10^{-5} for the MSE and under 1.6×10^{-4} for the PSNR. The software reconstruction had a duration of 453 minutes per iteration, while the GPU version completed an iteration step in 44 minutes resulting in a factor 10.3 speed increase.

5. DISCUSSION

5.1. Accuracy

The 6800 ultra GPU stores 32 bit floating point values internally using the IEEE standard 754 floating point format. However, not all GPU instructions are as accurate as their CPU counterpart. Library functions such as the sine and cosine are approximated and rounding errors might occur in an intermediate step before storing the result in the standard floating point format. Longer instructions such as 'multiply and add' are often subjected to these errors. The reconstruction algorithm however mainly depends on transformations of pixel positions and thus rotations which are

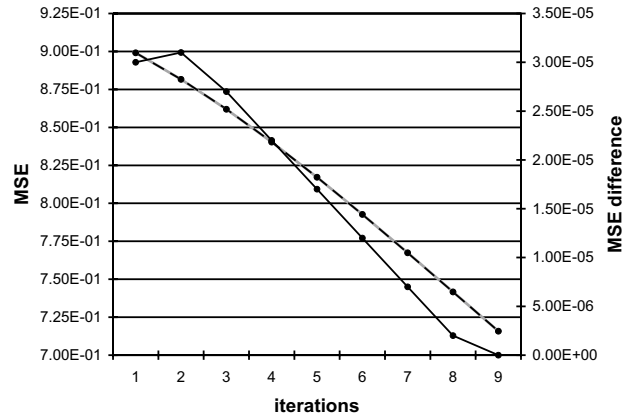


Fig. 15. Left axis: MSE of the GPU (black dashed line) and CPU (grey line) reconstruction compared to the analytical phantom. Right axis: difference values (black line)

reasonably accurate. The resulting floating point value is then floored to an integer value in order to determine the rows and columns as the index into the matrices. Therefore a small error could occasionally result in a different row or column than would be the case for the software algorithm. The software versus hardware comparison shows that although the RMS value slightly increases per iteration number for the reconstructed image, the error is insignificantly small. Moreover, as shown by the comparison to the analytical phantom this error does not result in a less accurate approximation per iteration step when compared to the software solution. The difference images illustrate that the spread of the error follows the spread of the counts in the image. We can conclude that the GPU reconstruction provides nearly equal accuracy compared to the software algorithm.

5.2. Speed

The reconstruction data is kept inside GPU memory for the duration of forward projection and the backprojection steps. Although it is possible to keep all data in GPU memory during the whole reconstruction it is not always beneficial to do so. The 2 seconds delay introduced by downloading a memory buffer from the GPU to main memory is insignificant compared to the duration of the forward projection and backprojection steps while a different and more suitable data format can decrease the bandwidth usage of the next step. This is obvious for the backprojection, in which the reduced system matrix was replaced by an orthogonal version. This way only 5 texture values (each with 4 floats) need to be read instead of 20 (consisting of 1 float). The speed increase of 10.3 was achieved by exploiting the par-

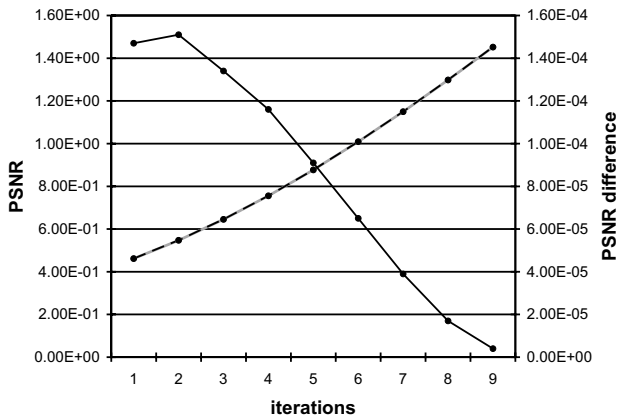


Fig. 16. Left axis: PSNR of the GPU (black dashed line) and CPU (grey line) reconstruction compared to the analytical phantom. Right axis: difference values (black line)

allelism of the GPU through the design of the algorithm.

6. CONCLUSION

This paper has focused on the implementation of a 3D reconstruction algorithm for rotating slat collimators using graphics hardware. The results showed a 10-fold speed increase, without making any compromise regarding accuracy. The algorithm was transformed into a GPU streaming model in order to exploit the inherent parallelism of graphics hardware. The graphics card used in this paper is a sub \$400, commercially available solution which is able to outperform recent CPU's while maintaining the same level of accuracy.

7. REFERENCES

- [1] S. Vandenberghe, R. Van Holen, S. Staelens and I. Lemahieu, *System characteristics of SPECT with a slat collimated strip detector*, Phys. Med. Biol., vol. 51, pp. 391-405, 2006.
- [2] T. R. Miller and J. W. Wallis, *Fast maximum-likelihood reconstruction*, J. Nucl. Med., vol. 33(9), pp. 1710-1711, 1992.
- [3] R. Van Holen, S. Vandenberghe, S. Staelens, Y. D'Asseler, I. Lemahieu, *Contrast noise behaviour for a rotating slat collimated gamma camera*, Accepted for poster presentation at EUROMEDIM Conference, Marseille, France, 2006.
- [4] Fang X., Mueller K., *Accelerating Popular Tomographic Reconstruction Algorithms On Commodity PC*

Graphics Hardware, To appear in IEEE Trans. Nucl. Sci.

- [5] A. Anderson, A. Kak, *Simultaneous Algebraic Reconstruction Technique (SART): a superior implementation of the ART algorithm*, Ultrason. Img., vol. 6, pp. 81-94, 1984.
- [6] H. Hudson, R. Larkin, *Accelerated Image Reconstruction Using Ordered Subsets of Projection Data*, IEEE Trans. Medical Imaging, vol. 13, pp. 601-609, 1994.
- [7] Fang X., Mueller K., *Ultra-fast 3D filtered backprojection on commodity graphics hardware*, IEEE International Symposium on Biomedical Imaging, ISBI 2004, Arlington, VA, april 2004.
- [8] Z. Wang, T. Li, G. Han, Z. Liang, *PC Graphics Card-Based Acceleration For OS-EM Image Reconstruction Of Quantitative SPECT With Varying Focal-Length Fan-Beam Collimators*, Proceedings of the Society of Nuclear Medicine, 51st Annual Meeting, Philadelphia, PA june 19-23, 2004.