Projector and Backprojector for Iterative CT Reconstruction with Blobs using CUDA

Rolf-Dieter Bippus[1], Thomas Köhler[1], Frank Bergner[1], Bernhard Brendel[1], Eberhard Hansis[2] and Roland Proksa[1]

Abstract— Using blobs allows modeling the CT system's geometry more correctly within an iterative reconstruction framework. However their application comes with an increased computational demand. This led us to use blobs for image representation and a dedicated GPU hardware implementation to counteract their computational demand. Making extensive use of the texture interpolation capabilities of CUDA and implementing an asymmetric projector/backprojector pair we achieve reasonable processing times and good system modeling at the same time.

Index Terms—Nvidia CUDA, blob driven projection, ray driven projection, interpolation, beam shape model.

I. INTRODUCTION

The aim of this work is twofold. We aim at modeling the CT system's geometry correctly eventually accounting for the more subtle effects influencing image quality and resolution. And we want to achieve this within reasonable processing times for iterative reconstruction algorithms such as SART [1] or Maximum Likelihood [2].

We decided to use blobs as proposed in [3] which, apart from a favorable image representation, allow to model major effects independent of the projection angle. An efficient blob-driven projector/backprojector that correctly models divergent beam geometry and finite detector pixels has been described in detail in [4] for a standard CPU implementation.

Most approaches described in literature utilizing graphics hardware use ray or voxel-driven approaches and make use of nearest neighbor or linear interpolation to calculate the image or projection values at dedicated positions [5][6][7]. Only few literature can be found on GPU acceleration using blob-based image representations such as [8], which uses a blob-driven approach for both forward- and backprojection. In order to avoid synchronization effort for write operations (e.g. atomic CUDA operations [8]) due to overlapping blob footprints on the detector we extend the approach taken in [4] by a ray-driven forward projection algorithm.

II. METHODS

A. System Geometry

As implemented in most commercial CT systems, this work is based on a focus centered, cylindrical detector as shown in Fig. 1. The detector is parallel to the rotation axis *z* and the transaxial plane is defined by the *x* and *y* axis. Fan angle Θ_β, cone angle Θ_γ and detector coordinate system *(u,v)* are shown. For the approximations used below, Θ_γ is assumed to be small.

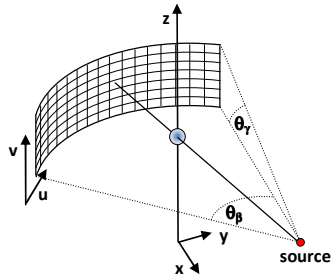


Fig. 1 CT system geometry with focus centered, cylindrical detector, fan angle θ_β and cone angle θ_γ.

B. Forward Projection

For forward projection we use a ray driven algorithm since this avoids synchronization effort for write operations due to overlapping blob footprints on the detector. In order to evaluate the line integral we need to calculate the contribution of all blobs to the beam and follow the scheme first introduced in [9] and extended to blobs in [10]. The blobs' contributions are evaluated slice-by-slice along the fastest changing transaxial slice index, *x* or *y*. This can be interpreted as sampling the image along the ray at each slice intersection using an appropriate interpolation kernel as shown in Fig. 2. In all of the following *x* and *y* can to be interchanged if the main direction of the ray is in *x* instead of *y*.

Fig. 3 shows the local beam geometry for a divergent beam traversing a blob at distance d to its center. If beam divergence is neglected locally (within the neighborhood of 4x4 grid points), the blob's contribution to the ray can be expressed in terms of distances d_u, d_v magnification $m = d_{SD}/d_{SB}$, angle γ and footprint $F(d) = \int_{-\infty}^{\infty} b(d,0,z)dz$ of a spherically symmetric basis function $b(x,y,z)$ and some normalization factor *K* (symbols $d_{SD}, d_{SB}, e_u, e_v, \gamma$ as defined in Fig. 3):

$$w = \frac{1}{K}\int_{d_u-D_u/2}^{d_u+D_u/2}\int_{d_v-D_v/2}^{d_v+D_v/2} F(\sqrt{u^2+v^2})dvdu$$
$$D_u = \frac{e_u}{m} \;;\; D_v = \frac{e_v}{m}\cos\gamma \qquad (1)$$

In order to calculate the interpolation weights for the 16 nearest neighbors to the point *P* at which the ray intersects a

[1]Philips Research Laboratories, Röntgenstraße 24, D-22335 Hamburg, Germany.
[2]Philips Healthcare, Nuclear Medicine, San Jose, CA
Corresponding author: rolf.bippus@philips.com

slice, the distances d_{ij} need to be calculated according to the geometry in Fig. 4 (γ and β as defined in Fig. 4):

$$\begin{aligned} d_{ij}^2 &= (x_p - x_i)^2 \left(1 - r_x^2\right) + (z_p - z_j)^2 \left(1 - r_z^2\right) \\ &\quad - 2 r_x r_z \left(x_p - x_i\right) \left(z_p - z_j\right) \\ r_x &= \sin(\beta) \cdot \cos(\gamma) \;\; ; \;\; r_z = \sin(\gamma) \end{aligned} \tag{2}$$

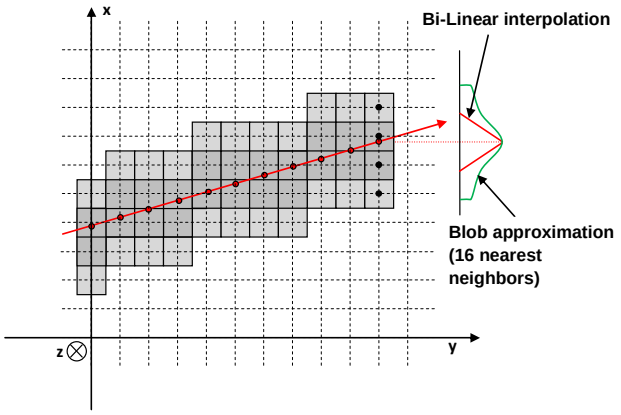


Fig. 2 Slice by slice sampling along the fastest index changing direction of the ray shown in 2D. For 3-D the slices for 2-D interpolation are perpendicular to the drawing plane extending into z-direction as indicated.

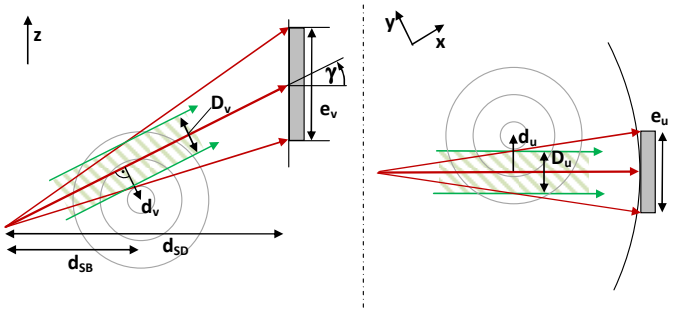


Fig. 3 Beam geometry for calculation of the mean contribution of a blob to a given ray at distance d by integration over the sensitive detector area for focus geometry with cylindrical detector. The beam is locally approximated by a beam of parallel rays (green shaded area).

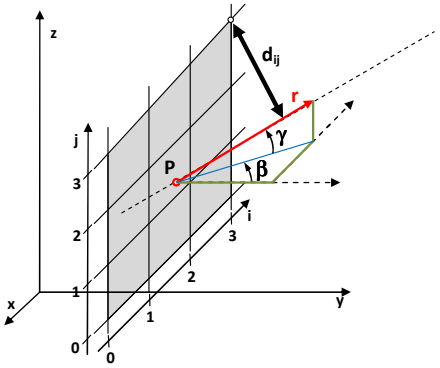


Fig. 4 Interpolation at intersection $P = (x_p, y_p, z_p)$ of the ray with a slice. Distance d_{ij} of a blob center at grid point (x_i, y_p, z_j) to the ray with direction $\vec{r} = (r_x, r_y, r_z)$.

For an efficient implementation we want separable interpolation kernels. However due to the mixed term in d_{ij} even for separable blob footprints the resulting weights will in general not be separable in *i* and *j*. If however the cone angle Θ_γ is small and so are the possible values for γ we can approximate $r_z \approx 0$ and $r_x \approx \cos(\beta)$ (coordinates *x*, *y*, *z* normalized to grid spacing):

$$\begin{aligned} d_{ij}^2 &\approx (x_p - x_i)^2 \left(1 - r_x^2\right) + (z_p - z_j)^2 \\ &\approx (x_p - x_i)^2 \cdot \cos^2(\beta) + (z_p - z_j)^2 \\ &= d_u^2(i, x_p - \lfloor x_p \rfloor, \beta) + d_v^2(j, z_p - \lfloor z_p \rfloor) \end{aligned} \tag{3}$$

with implicitly defined d_u depending on *i* and d_v depending on *j*. Inserting d_u and d_v into eqn. (1) and using a separable footprint $F(\sqrt{u^2 + v^2}) = \phi(u) \cdot \phi(v)$ we get an interpolation kernel separable in *i* and *j*:

$$\begin{aligned} w_{ij} &\approx \frac{1}{K} \left[\int_{d_u(i) - D_u/2}^{d_u(i) + D_u/2} \phi(u) du \right] \left[\int_{d_v(j) - D_v/2}^{d_v(j) + D_v/2} \phi(v) dv \right] \\ &= w_i(x_p - \lfloor x_p \rfloor, \beta, m) \cdot w_j(z_p - \lfloor z_p \rfloor, m, \gamma) \end{aligned} \tag{4}$$

C. Backprojection

The backprojection is based on the blob-driven approach described in detail in [4]. Fig. 5 shows the geometry to calculate the necessary footprint on the detector.

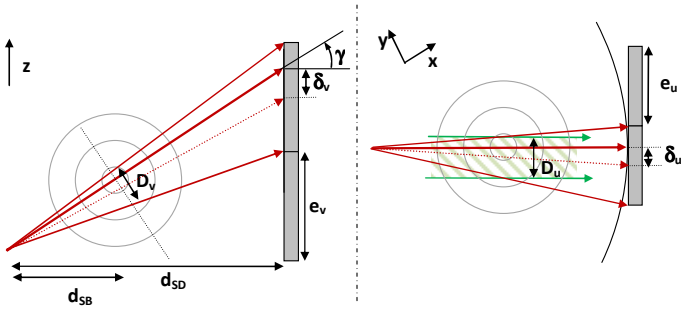


Fig. 5 Beam geometry for calculation of the blob footprint on the detector.

The ray through the center of the blob hits the detector at position (u_p, u_v) at distances δ_u and δ_v to the nearest detector pixel center. The weight (footprint) for one detector element can be again calculated via eqn. (1). If the angle γ is assumed approximately constant over all rays passing through the blob the 4x4 detector footprint can be calculated by

$$\begin{aligned} w_{ij} &\approx \frac{1}{K} \left[\int_{d_u(i) - D_u/2}^{d_u(i) + D_u/2} \phi(u) du \right] \left[\int_{d_v(j) - D_v/2}^{d_v(j) + D_v/2} \phi(v) dv \right] \\ &= w_i(\delta_u, m) \cdot w_j(\delta_v, m, \gamma) \\ d_u(i) &= \frac{1}{m} \left(u_p - \delta_u + (i - 1) \cdot e_u\right) \quad \forall\, 0 \le i \le 3 \\ d_v(j) &= \frac{1}{m} \left(v_p - \delta_v + (j - 1) \cdot e_v\right) \quad \forall\, 0 \le j \le 3 \end{aligned} \tag{5}$$

In contrast to the forward projection there is no ray-direction dependent tilt (the angle β in Fig. 4) since the detector is focus centered. A direct consequence is that no additional approximations have to be made to ensure separability of footprints [4].

D. Implementation

The implementation makes use of the multi-processing capabilities of the graphics board, as opposed to adapting the problem to the accelerated graphics pipeline [5]. Using the native CT geometry, each thread thus calculates the direction of the ray hitting a detector bin (during forward projection) or the mapping of the image grid point onto the detector (in backprojection).

For separability we use Gaussian kernels for image representations instead of Kaiser-Bessel functions. The optimal Kaiser-Bessel function found in [11] has order 2, $\alpha = 10.4$ and radius 2. This is well approximated by a Gaussian with standard deviation $\sigma = 0.53$. Where necessary the calculated footprints are cut-off to the 4x4 interpolation kernel and re-normalized to the sum of the full footprint.

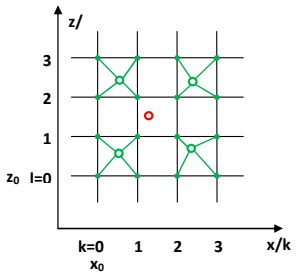


Fig. 6 4x4 separable interpolation using 4 bi-linear interpolations.

A 4x4 interpolation with separable weights

$$s = \sum_{l=0}^{3}\sum_{k=0}^{3} w_{kl} f(x_0 + k, z_0 + l) \qquad (6)$$
$$w_{kl} = w_k^x \cdot w_l^z$$

can be calculated via 4 bilinear interpolations (Fig. 6). If $f(x, y)$ denotes the image values and the bilinear interpolation at $(x + \Delta_1, y + \Delta_2)$ is denoted by $f_l(x + \Delta_1, y + \Delta_2)$ with $0 \le \Delta_1, \Delta_2 < 1$, we obtain

$$\begin{aligned} s = k_1^z \cdot [\quad & k_1^x \cdot f_l(x_0 + \Delta_1^x, z_0 + \Delta_1^z) \\ & + k_2^z \cdot f_l(x_0 + \Delta_2^x, z_0 + \Delta_1^z)] + \\ k_2^z \cdot [\quad & k_1^x \cdot f_l(x_0 + \Delta_1^x, z_0 + \Delta_2^z) \\ & + k_2^x \cdot f_l(x_0 + \Delta_2^x, z_0 + \Delta_2^z)] \end{aligned}$$

$$k_1^x = w_1^x + w_0^x \;\; ; \;\; \Delta_1^x = \frac{w_1^x}{w_1^x + w_0^x} \qquad (7)$$
$$k_2^x = w_2^x + w_3^x \;\; ; \;\; \Delta_2^x = \frac{w_3^x}{w_2^x + w_3^x} + 2$$
$$k_1^z = w_0^z + w_1^z \;\; ; \;\; \Delta_1^z = \frac{w_1^z}{w_1^z + w_0^z}$$
$$k_2^z = w_2^z + w_3^z \;\; ; \;\; \Delta_2^z = \frac{w_3^z}{w_2^z + w_3^z} + 2$$

The weights needed to perform fast 4x4 interpolation (eqn. (7)) are precalculated into lookup tables which are accessed via texture mapping and bi-/tri-linear interpolation depending on the number of free parameters. The capability of CUDA (Nvidia Corp, Santa Clara, CA) to have textures of 4 channels allows to store the weights $k_{1,2}$ and $\Delta_{1,2}$ for each dimension *x* and *z* into a single texture.

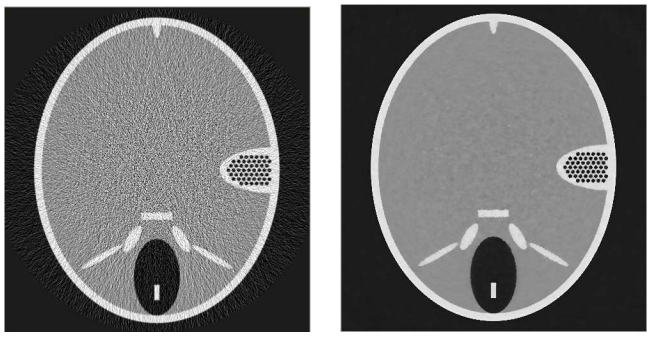
Due to the ray-driven forward- and blob-driven backprojection write operations to the same memory location by different threads are completely avoided. During forward projection the projection data is segmented into tiles and each tile is processed by a single block of threads. Additionally each ray is sampled by multiple threads with the result being accumulated in shared memory. The final projections are updated after all threads of a block have finished. For backprojection each thread operates on multiple image grid points. The threads of each block process image grid points in an interleaved fashion such that the write operations to the image occur approximately simultaneously to contiguous memory locations. The block sizes have been empirically optimized. Image and projection read-access are realized by mapping the memory to 3D and 2D textures utilizing bi-linear interpolation via a single lookup.

E. Linear interpolation

For comparison the projector and backprojector have also been implemented using bi-linear interpolation. In this case no footprints need to be calculated. The basic kernel routines are identical except that interpolation is performed by accessing the corresponding textures (image slice or projection data) directly via bi-linear interpolation.

III. Evaluation

A. Simulation / Reconstruction



(a) (b)

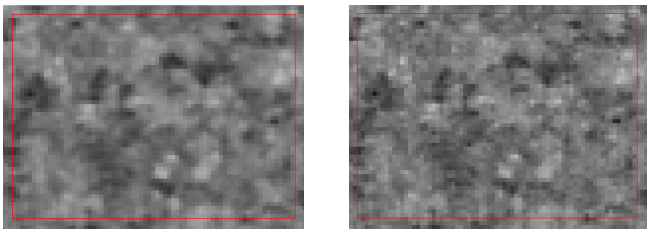
Fig. 7. Reconstructions of simulated FORBILD head phantom, central slice. (a) FBP with ramp filter. (b) Iterative ML reconstruction with Huber penalty, δ=16HU, β=0.3. Level 50 HU, window 2000 HU

The FORBILD head phantom has been analytically simulated with a distance of the source/detector from the rotation axis of 570/470 mm, with a detector of 512 x 64 square pixels of edge size 0.91 mm and with circular trajectory. Matching image size was 512x512x81 with a cubic grid spacing of 0.5 x 0.5 x 0.5 mm. The presented forward and back-projection have been embedded into an SPS iterative reconstruction [2] using a Gaussian noise model for line integral measurements. 100 iterations and Huber regularization with δ=16HU are used. Poison noise has been added to the projections. Fig. 7 shows the central slice from sample reconstructions.

B. Image Quality

Fig. 8 and 9 show comparisons for the blob interpolation using 4x4 footprints and the linear interpolation. In order to achieve comparable reconstructions, the regularization weight β was tuned to obtain the same background SNR level in both images. This meant using a slightly stronger regularization for linear interpolation due to the additional noise being

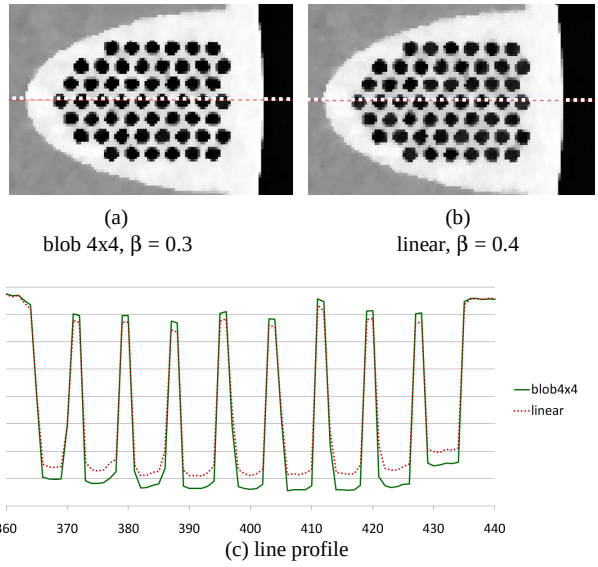
introduced. Fig. 8 shows the smoother background noise structure obtained with blob interpolation and less regularization, Fig. 9 the enhanced resolution in the fine structures of the phantom when using blobs.



(a) blob 4x4, β = 0.3 SNR_{BG} = 30.7

(b) linear, β = 0.4 SNR_{BG} = 30.4

Fig. 8. Different noise structures at center of rotation for blobs and linear interpolation. Regularization parameters chosen for matched background SNR. Level 50 HU, window 300 HU



(a) blob 4x4, β = 0.3

(b) linear, β = 0.4

(c) line profile

Fig. 9. Line profiles through fine structures of the central slice. Comparison of blob and linear interpolation. Regularization parameters chosen for matched background SNR.

C. *Timing*

The processing times shown in table 1 have been measured on an Nvidia Tesla C2050 board with 3 GB total memory and 14 multiprocessors and ECC (Error Correction Code) enabled. The times have been measured for 512 projections of 128x512 detector bins. The fan covered the full image width at the center of rotation. Image slices were 512^2 and a total of 266 slices were covered by the cone for a circular trajectory and thus updated in BP. The times shown are pure GPU kernel runtimes, memory transfers are not included.

IV. Discussion

The timings for linear interpolation compare well to what is found in literature taking into account the fact that we use native system geometry in order to avoid any re-binning steps. This might come with some loss in potential processing speed, since planar projections allow further speedup as successfully shown in [5]. Nevertheless the method described here shows that a 4x4 blob footprint can be used resulting in acceptable

Table 1

Processing times for projections on an Nvidia Tesla C2050 board for 512 projections of 128x512 each. The image volume consisted of slices of size 512^2 with 266 slices within the FOV.

forward proj.	linear	2.3 s
	blob 4x4	8.6 s
backprojection	linear	3.3 s
	blob 4x4	5.4 s

overhead. The higher impact on the forward projection can be explained by the fact that for a ray-driven projection the geometry is calculated only once per ray and thus sampling (interpolation) along the ray is more dominant compared to the voxel/blob-driven backprojection with a geometry calculation for each image grid point. In addition there is a reduced locality of interpolations in the volume (for FP) as compared to interpolation in the 2D projection views (for BP).

We conclude from the above results that using GPUs and adequate implementations of the projectors, iterative reconstruction using blobs for image representation becomes feasible. This, along with avoiding re-sampling, will allow applying detailed system modeling for enhanced resolution/noise tradeoff. Strictly speaking the method presented here results in an asymmetric projector-backprojector pair, mainly due to the limited footprint used in projection and image space. The asymmetries increase with an increasing mismatch in projection and image sample spacing. This effect will need close attention.

References

[1] A. Andersen and A. Kak, "Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm" in Ultrason. Imaging 6, 1984, pp 81-94

[2] J. A. Fessler, "Statistical image reconstruction methods for transmission tomography" in Handbook of Medical Imaging,, J.M. Fitzpatrick and M. Sonka (ed.) (SPIE, Bellingham, WA, 2000), vol. 2, chapter 1, pp. 1-70

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

[4] A. Ziegler, T. Köhler, T. Nielsen and R. Proksa, "Efficient projection and backprojection scheme for spherically symmetric basis functions in divergent beam geometry" in Medical Physics 33(12), December 2006, pp. 4653-4663

[5] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware" in Physics in Medicine and Biology 52(2007), pp. 3405-3419

[6] D. Vintache, B. Humbert and D. Brasse, "Iterative Reconstruction for Transmission Tomography on GPU using Nvidia CUDA" in Proc. Intl. Meeting on Fully 3D Image Reconstruction 2009, Workshop on High Performance Image Reconstruction, pp. 78-81

[7] M. Kachelrieß, M. Knaup and O. Bockenbach, "Hyperfast parallel cone-beam backprojection using the cell general purpose architecture" in Medical Physics, 34(4), April 2007, pp. 1474–1486.

[8] H. Wang, L. Desbat and S. Legoupil, "2D X-Ray CT Reconstruction Based On TV Minimization and Blob Representation" in Proc. Intern. Conf. on Image Formation in X-Ray Computed Tomography, 2010, pp. 252-255

[9] P. M. Joseph, "An Improved Algorithm for Reprojecting Rays Through Pixel Images" in Trans. on Medical Imaging, Volume MI-1, no. 3, Nov. 1982, pp 192-196

[10] L. M. Popescu and R. M. Lewitt, "Ray tracing through a grid of blobs" in Nuclear Science Symposium Conference Record, 2004 IEEE, Vol. 6, pp 3983 – 3986

[11] S. Matej and Robert M. Lewitt, "Practical Considerations for 3-D Image Reconstruction Using Spherically Symetric Volume Elements" in IEEE Trans. on Medical Imaging, 1996, vol. 15, no. 1, pp. 68-78